

3주차 강의영상

Community의 특징 : 상시적, 가로수형, 통합적

Open-Source SW : 주체, 이익, 부작용

۶

21

「개발자」 ⇒ 실무 인정받는게 중요 . Open-Source SW가 기억

⇒ 74 될 수 있는 Public domain = 07쪽

강의 Motivation \Rightarrow drive

작곡작연 이익

↳ 지역기술인 발전

7월 - 7월 Win-win Model

$\Gamma_1 \vdash C, UNLx$ 증명해. 고급언어 \rightarrow library

기호자수작

↳ 1980년대 실리콘밸리의 혁명!

1990년대 인터넷 → dot Com (.Com) 붐↑

→ 7장 7주차 1, Red Option

『생산성 11 & 안정성 11 & 비용 11』

UX, Brand, Lock-in

v

단위수가 다 많으니 기능이 늘친다. $\rightarrow UX, UI$ 등의 중요도↑

feature, function

기능

→ 유지보수, 흐름화

시끄러운 불온감과 위축과 나온 : 기관고사 + 철수적인 내용

마지막 주자 노 (2월 21일 수업과 나온)

1, 2주차가 중요.

이번 OS가 제일 높아 나오고 2학기 학점이 뜰, 예상 가능성이 있음.

D4: 유닉스 / 리눅스 + 명령어, 쉘

Open Source SW
사용자

=
유닉스

o Why Linux?

- 유닉스/리눅스 OS : 1970년 AT&T에서 연구소에서 개발 - 자유 라이선스

스마트폰, PC, 서버, 스마트워치

SW 경쟁력 향상

less

- 유닉스/리눅스 기반 운영체제

- 작업 관리 -

. 안드로이드 OS

. iOS

. Mac OS X

. 리눅스 (Linux)

. BSD 유닉스

. IBM AIX, ...etc

Windows Mobile, Palm (안드로이드 X)

(=> DOS, Windows (아예 X))

최소화된 환경, 자원에 대한 영향 최소화

- o 유닉스 설계 원칙: 단순성, 일관성, 개방성, 소스코드 개방성, 라이선스 ...

C언어 작성.

일관성 ↑

o 유닉스 특징

□ 다중 사용자, 다중 프로세스: 여러 사용자, 여러 프로그램 동시에 작동.

관리자 혹은 슈퍼유저가 있음
(root, root는) (user, 사용권한)

o Shell (쉘) 프로그래밍 (Scripting)

명령어, 유형언어 등을 사용하여 자동화한 프로그램.

□ 네트워킹: 유닉스의 기본 네트워킹 시스템

4-2: 유닉스 시스템 구조 · 안정성↑

○ 유닉스 운영체제 구조

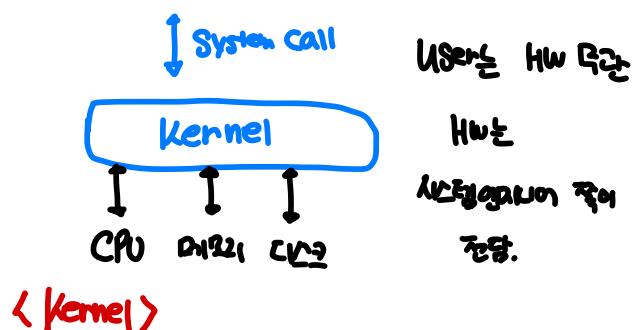
○ 운영체제: 컴퓨터 HW 자원 운영, 관리
프로그램 실행할 수 있는 환경 제공

○ 커널 (Kernel): 운영체제 핵심. HW 운영 및 관리

○ System Call: 커널이 제공하는 서비스에 대한
프로그램의 인터페이스 역할

○ Shell: 사용자 $\xrightarrow{\text{사용자 인터페이스}}$ 운영체제
사용자 $\xrightarrow{\text{명령어}}$ Shell이 해석 및 수행
(명령어 해석기)

■ Kernel: 프로세스, 파일, 메모리, 통신, 주변장치 등



◆ 1 Process 관리

▶ 프로세스가 실행할 수 있도록

▶ 프로세스들을 CPU 스케줄링하여 꾸준히 수행

1 파일 관리

▶ 디스크 같은 저장장치에 파일 시스템 구조

2 파일 관리

1. 메모리 관리: 메인 메모리 할당과回收 관리

2. 통신 관리: 네트워크 통해 정밀 디버깅을 수 있도록 관리

3. 주변장치 관리: 모니터, 키보드 등 사용 가능한 장치 관리

4.3 유닉스 역사 및 버전

시작: Open UC → AT&T : Close. 상용화 이용 → Open (Linux)

유닉스 다양한 종류만들 뉴클레리온
글쓰기

① 시작: Ken Thompson이 아침불씨에로 개발

D. Ritchie C언어 사용

유닉스작성으로 만든 OS로 언어

이종점으로 C 험파워로만 있으면 어려가능.

소스코드 대량 개방

② 유닉스 큰 흐름 [유닉스 버전으로 묶음]

① 시스템/V : AT&T가 오스스 C언어. 라이선스화

② BSD 유닉스 : Open. 뉴클레리온 BSD 키워

③ 리눅스 (Linux) → From Scratch
(새로만들기)

리눅스 : PC를 위한 효율적인 유닉스 시스템

· 소스코드 공개 → Community 활성화 → Open Source SW 도착

④ 인터넷자료들을 이 유틈 가능성이 및 확장

중庸 도메인상의 유틈 OS

· HW, Platform 자료도, 성능↑, 안정성↑

· GNU 소프웨어와 함께 배포

기본 명령어 : Shell을 통한

○ 시간확인

\$ date : 날짜 및 시간확인

<사람문제>

명령어 : 예술

○ 시스템정보확인

\$ host name · 등록된 host name

① 실제 실행 → 시작이 어렵.

② 약어 암기

\$ uname : Linux (시스템이 쓰고 있는 유닉스 이름)

\$ uname -a : Linux 유저명 ~ (↑ 자세한 이름)

○ 사용자정보확인

\$ whoami : 로그인된 계정명

\$ who : 이 시스템에 누가 있다

※※

○ 디렉토리 내용확인

\$ ls : 디렉토리의 어떤 파일인지 리스트 알려주기

○ Pw 번호

○ 화면정지

\$ passwd

\$ clear

※※

○ Dnf(혹은 manual) 명령어 사용법

\$ man ls · 메뉴얼 리스트.

\$ man : 메뉴얼. Page 넘기 가능.

○ 기본 명령어 사용

\$ whatis ls : 특징 명령어 확인 가능

what + 명령어.

명령어 노명 뜻

Shell (쉘)

• What is Shell?

DOS의 계승기

사용자 ↔ 운영체제 사이 창구역할을 하는 SW

• Shell의 종류

유닉스/리눅스에서 사용가능한 Shell

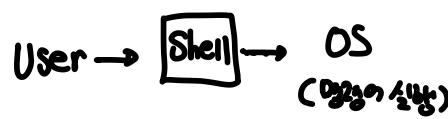
Bourne Shell

Korn Shell

C Shell : Shell 기능 + C언어특징

Bash Shell : unix CloseToK, 디시언트 (Bourne Shell 계승으로 푸아K, 만든다)

tCsh Shell



• login Shell

로그인하면 자동으로 실행되는 Shell

보통 시스템 관리자가 설정 만들 때 자동

www www /bin/bash

• 기능

DOS의 계승

시작파일

스크립트 (Shell 자체 내부 언어)

• 실행절차

↓
시작파일 읽고 실행

↓
프로토콜 출력, 사용자 명령 대기

↓ fork

사용자 명령 실행

↓ Ctrl+D

종료

• Shell의 환경변수

• 환경변수 설정법

\$ 환경변수명 = 문자열

환경변수의 값을 문자열로 설정

\$ TERM=xterm : 설정

\$ echo \$TERM : 확인

xterm : 출력 (확인)

• 사용자 정의 환경변수

\$ MESSAGE=Hello

\$ export MESSAGE

export 된 사용 가능 (환경변수로)

• 환경변수 초기화

\$env

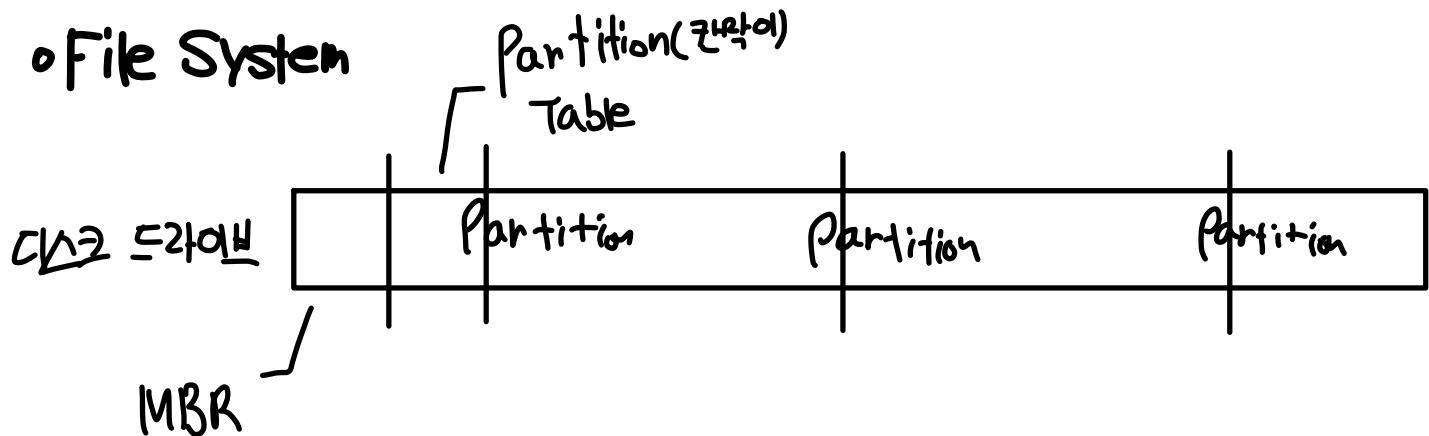
05. File

- File: 정보를 저장, 탐색, 노드, 자식, ..

저장장치 파일(Paper ~ 마그넷 테이프 → SSD ...) 뿐만 아니라 물질은 출처는 출처

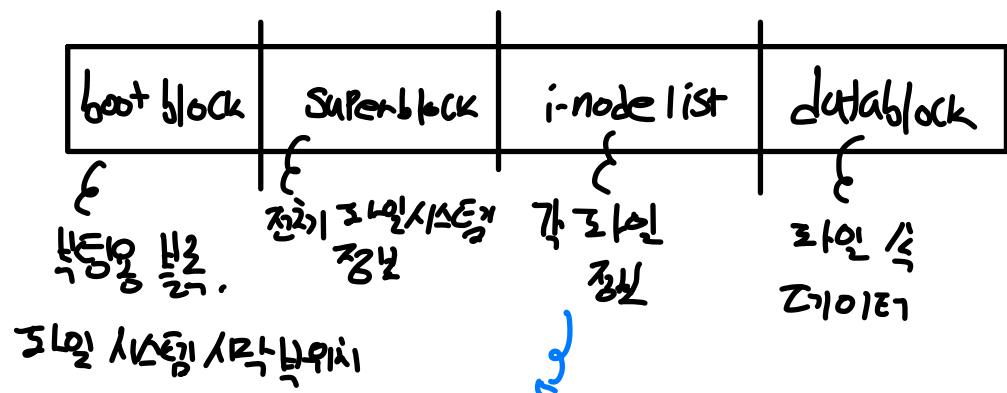
~~File~~.

- File System



정해진 위치에 항상 같은 값이 있음.

하위 디스크 활용

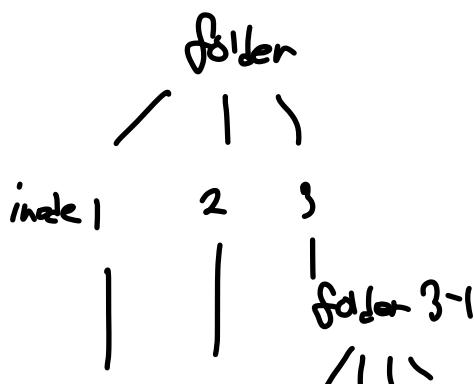


Folder (= 디렉토리 파일)

: i-node 번호 + 이를 가리kan 파일

(Folder → i-node 속 있는 파일)

파일구조



일반파일 : exe, obj, dll(아이디) ...

폴더 : folder

장치파일 : 키보드 등 물리적 장치 내부로 포함

심볼릭 링크 파일 : 어떤 파일에 대한 링크로 정의된 파일

디렉토리 명령어

0 디렉토리 2스트

\$ ls [옵션] [파일 / 디렉토리] : 지정된 디렉토리 내용 2스트

자장수시 헌재 디렉토리 내용 2스트

파일 자장수 2스트

[옵션]

-a : 숨겨진 파일 포함 모든 파일 2스트 (all) : ls -a

-S : 파일의 크기 K바이트 단위로 출력 (Size)

-l : 파일의 상세정보 출력 (long)

-F : 파일의 종류를 표시하여 출력

-R : 모든 하위 디렉토리를 2스트

-Sl : size + long
ex) \$ ls -Sl CS1.txt

-aSl은 a+Sl. 모든 파일 Sl

4 -rw-r--r-- 1 ywcho CS 2088 4월 16일 13:37 CS1.txt

① 파일크기 ④ 권한 ⑦ 파일명

② 파일종류 ⑤ 사용자ID ⑧ 처리수정시간

③ 접근권한 ⑥ 그룹ID ⑨ 파일명

~~접근권한~~ 접근권한

- : 일자 r : 블록디바이스 1: 링크
w : 디렉토리 c : 문자디바이스

	File	Directory	숫자형	소유자 / 그룹 / 기타로 구분하여 관리
r:	파일 읽기권한	디렉토리 파일을 읽을수있음 : 2 ⁰	4	소유자 그룹 기타 사용자 권한
w:	쓰기권한	파일 생성 or 삭제권한 : 2 ¹	2	r r w w
x:	실행권한	파일 실행권한 : 2 ²	1	- - x x

2^{0+2^{1+2²}} 2^{1+2²} 2^{2+2⁰}

7 5 5

○ 현재 디렉토리 출력

\$ Pwd (Print Working Directory)

/home/yahoo : 현재 작업 디렉토리 절대 경로명 출력

○ 디렉토리 이동

\$ Cd [디렉토리] : 현재 작업 디렉토리를 지정한 디렉토리로 이동.

ex) \$ Cd Desktop 파일胥이 흥 디렉토리로 이동

○ 명령어의 경로 확인

\$ Which [명령어] : 명령어의 절대 경로를 보여줌

ex) \$ Which ls

/bin/ls : 출력

○ 디렉토리 생성 : make directory

\$ mkdir [-P] [디렉토리] : 디렉토리(들)을 새로 만듬

\$ mkdir test

\$ mkdir test2 temp & 2개 만들기

[-P]는 필요하지 않아도 디렉토리를 자동으로 만들어주는거.

mkdir dest/dirl : 오류. dest 없음

mkdir -P dest/dirl : 정상작동

○ 삭제 : remove directory

\$ rmdir [디렉토리] 주의*: 빈 디렉토리만 삭제 가능

파일 생성 및 편집

0 간단한 파일 만들기

\$ Vi [파일이름] : Linux 기반 파일 편집기

- File은 사용자가 편집하고자 하는 File로
- File은 저장하지 않고 시작 가능

④ 편집 버퍼에 키이터 넣을 때

1. 키이터 쓰고 싶은 곳으로 커서 이동
2. 입력 모드로 바꾸기 (i)
3. 키이터, 입력
4. 명령 모드로 바꾸기 (ESC)

④ Vi 명령어

종료 명령을 입력할 수 있는 명령 모드가 있어야 함.

:wq : 작업 내용 저장하고 종료

_____ :q! : // 하지 않고 종료

종료 시 데이터 저장 상태점검

mp :는 글로모드 :nq! :

_____ :q! : 가능.

Vi에서 사용할 수 있는 명령어들을
이용하는 곳

0 생/파일 출력 : Concatenate : 연결하다

\$ Cat [-n] [파일이름] : 파일(들)의 내용을 그대로 화면에 출력.

직접 터미널에 내용 //

\$ touch 파일 : 파일 크기가 0인 이름만 있는 빈 파일 생성 Com. 펌웨어 모듈 때

i) 파일명이 기존의 존재하는 이름이면] 날짜/날짜로 파일이
접근 시간을 현재 시간으로 바꿈] 수정된 날짜 출력.

\$ more [파일]: 파일의 내용을 Page 단위로 출력

\$ head [-n] 파일: 파일의 앞부분을 출력. 자동X시 표준입력 내용 대상

\$ tail [-n] 파일 : 1 뒷부분 //

몇줄인지, \$ head -5 CS3.txt

파일 사용

n 있으면 No Y 있으면 Yes 선택 가능

0 파일 복사: Copy

\$ Cp [-i] 파일 1 파일 2 : 파일 1을 파일 2에 복사

파일이 있는 경우. -i 없으면 파일 2 내용 있어도 덮어쓰기(Cover Write)
-i 있으면 No. 복사하지 않음

\$ Cp 파일 디렉토리: 파일을 지정한 디렉토리에 복사
↳ 디렉토리에 똑같은 파일 생성

\$ Cp 파일 ... 파일 n 디렉토리: 여러개도 가능

\$ Cp [-r] 디렉토리 1 디렉토리 2: 디렉토리 1 디렉토리 2에 복사

{
Recursion.

단위 디렉토리 포함 전체 디렉토리 복사

0 파일 이동: 대화형 응답. 덮어쓰지 않기 위해. 즉 이동의 같은 파일 있으면 덮어쓸지 유/무

\$ mv [-i] 파일 1 파일 2 : 파일 1의 이름을 파일 2로 변경

\$ mv 파일 디렉토리 : 파일을 지정한 디렉토리에 복사. 예외 가능

↳ 원래 있던 파일은 삭제됨! Cp 차이?

\$ mv 디렉토리 1 디렉토리 2 : 1 → 2로 이름 변경

0과 일상화 : Remove

\$ rm [-i] 파일 : 파일

\$ rm [-rf] 디렉토리: -r: 디렉토리 아래 모든 파일 삭제

-i : 대화중 읽기 : 나에게 있어 중요한 말씀이 있어서 읽어주는 것.

리그: In \rightarrow 디렉토리 내에서 추가하는 마인드

심불국 징코. 있으면 심불국, 없으면 아드레코

\$ ln [-s] 파일1 파일2 : 파일 1이 대상 새로운 이름(링크)로 파일 2를 만든다.

\$ \ln[-s] \text{ 도일} | \text{ 디렉터리: } \text{도일} | \text{이 대안 경로를 지정한 디렉터리, 같은 이름으로 만든다.}

CP2101: 1491 012 022H (In) VS 245 022H (CP)

○ 심볼릭 링크 : 다른 파일 가리키고 있는 별도의 파일 . 일종의 툴수 파일

(이름 → 윤봉) 이 파일이 다른 파일에 대한 간접적인 참조인데 윤봉
'로' 양이, 윤봉과의 이름을 가지는 것으로, (전체 다른 파일이므로 윤봉으로 명명한 이유) 윤봉

◦ 하드웨어 : 기존 파일의 대량 새로운 이를

서예가 향기

(여는 \rightarrow 닫음) 솔직히 기조, 도입을 대처하는 i-노드를 가리켜 구현.

즉 윤보자와 동일한 i-node. i-node 간에는 서로 다른 이름, 원인 개념X

- 파일 수정·삭제 -

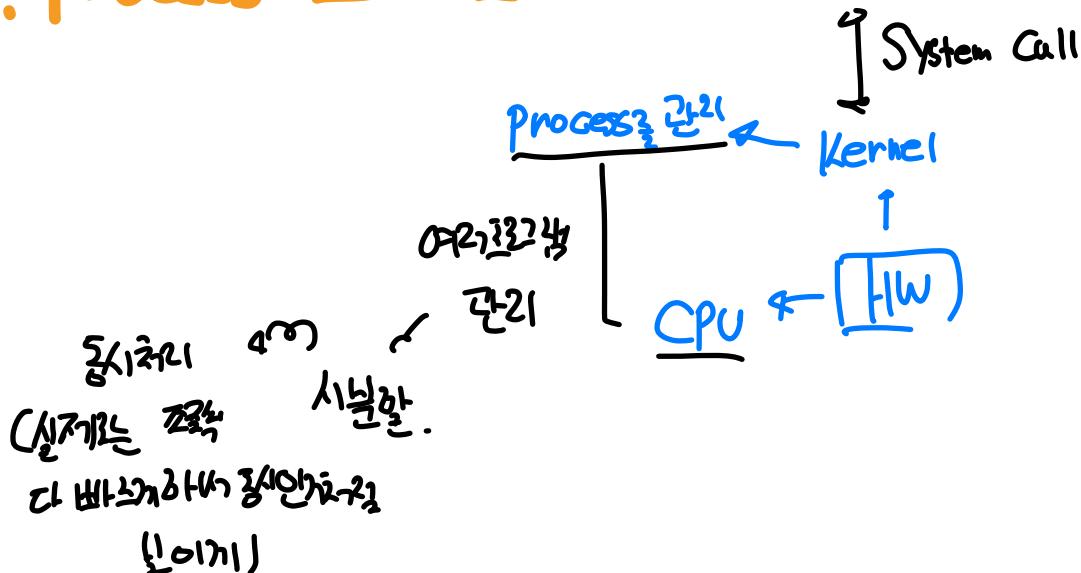
파드羸크: 원본 삭제 문구 X

심불의경: 윤보선제주시 윤보선
간드의를 암으면 안됨

06. Process - 프로세스

Shell

2014.2.21



○ What is Process?

프로세스의 instance

- 실행중인 프로세스는 Process 할. 「CPU 자원 배분 단위」
- PJD라는 프로세스 번호를 가짐. 필수적인 멤버는 프로세스는 고정된 PID 가짐.
- 각 프로세스는 부모 프로세스에 의해 생성 //하고 상위는 부팅

Process의 구조 (Process Kernel & Stack 구조)

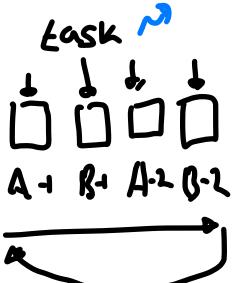
↳ Multi tasking

○ EHA (TASK) → 동시에 수행되는 것처럼 보이게 하는 방법

사람마다 서로 다른 단위.

ex)

구조체 단위.



사람은, 자료에서 빠간 단위.

Task Switching 하면 프로세스로 다음 task 이동 · 죽음
→ 같은 단위 task를 이동.

순차적

비행기一樣

- 프로세스 ID (PID) : 시스템 물론 X. 명령어 흔들기
다른 실행될 명령어 주소. fork - 자식 프로세스 탄생
두 프로세스
부모, Program Counter
동동이(모든자신)
fork 고정되어 함.
내용 많음. 나중에 풀어.
↓ PID, PFD간 다른
통도.
- 프로세스 구별용 고유값
 - Kernel 내의 process table의 index
 - 주로 커널 프로세스 : 수행순서를 늘 정해진 Process
- | | | | |
|--------------|------------------------------|-------|-------------------------|
| Booting
용 | 0 - Swapped
1 - init
⋮ | fixed | ① 사용 프로그램 실행(er: shell) |
|--------------|------------------------------|-------|-------------------------|
- fork() 1번출 2번 2회 런던
자식 : 0번은 /dev/mem - 1
부모 : 자식 PID 2번
- ② 동일한 프로그램 실행
같은 프로그램 불리기, 각각
다른 부모 실행
(ex: 네트워크 서비스)
- 자식, 부모 모두 되가 되거나 시작하지 않다면 모름.
- 프로세스 상태 보기 (Process Status)

\$ PS [-옵션] : 현재 시스템 내에 존재하는 프로세스들의 실행상태 요약, 출력.

- 등장 프로세스 리스트 : pgrep
- 걍 글자, 정규식, 등을
명령어 → 정규식으로 다양화 방식. 같은 글자
글자나 찾기 ~ ...
- \$ Pgrep [-옵션] [파인] : 프로세스 해당하는 프로세스들만 끌어온다.
- l : pid와 함께 프로세스의 이름 찾기 출력
 - n : 가장 최근 프로세스만 출력

정규식 (Regular Expression) : 사용 불편 X but 매우 유용한不過

· 유닉스 시스템에서 검색을 목적으로 활용

기타 문자를 통해 검색 효율적 가능

Path : 실행 파일까지 포함
경로명 /bin/ls

file : 실행 파일만 ls

작업 / 프로세스 관리

• 높은 프로세스

Shell

1. 프롬프트를 내고 대답이 입력 받음



2. 자식 프로세스 생성 fork 실행



3. 자식 프로세스에게 대답을 실행시킴 exec 실행

(시작하기)

자식 프로세스에게

사로잡기 시작

자식 프로세스에게

사로잡기 시작

점유하는 물리적인 대체되는 새 프로그램 main()

부터 시작

- 프로세스 상태 -

Suspend

↑ HUP, SSO..

CPU
사용

사용

fork → Create → Ready

Running

- Stopped -
- Traced -

끝나는
exit()만
남은 상태

Zombie

Delete

\$ (sleep 10; echo done)

[1] 8320

\$ kill 8320 or \$ kill %1

[1] Terminated (sleep 10; echo done)

• 프로세스 끝내기

(ex)

\$ kill pid

\$ kill %작업번호

: 프로세스 번호 (or 작업번호)가 대상하는 프로세스 강제종료

Echo [문자열] [문자열]: 문자열을 마지막에 '\n' 포함해서

준비한대로 출력하는 명령

• 프로세스 기다리기

\$ wait [pid]: pid로 지정한 자식프로세스가 종료되면 기다림. 다른 시스템에서 대기하는

ex \$ (sleep 10; echo 1번)

[1] 1231

\$ echo 2번 ; wait 1231; echo 3번

2번

3번

0 프로세스 우선순위 ↑ 기준값 0. 20 ~ 19 ↓

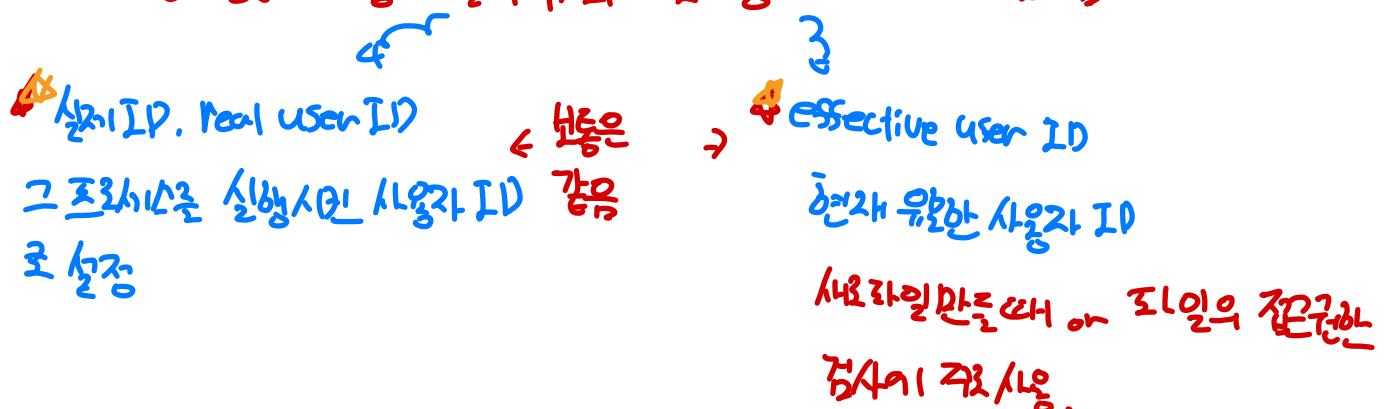
\$ nice [-n 조정값] 명령어 [인수들] : 주어진 명령을 조정한 우선순위로 실행

\$ renice [-n] 우선순위 [-gpu] PID : 이미 실행중인 프로세스의 우선순위를 다시한우선순위로

- g: 해당 그룹에 대한 모든 프로세스를 의미
- p: 해당 프로세스의 PID를 지정
- u: 지정한 사용자명으로 소유된 프로세스를 의미

프로세스, 사용자ID: UID

\$ id [사용자명]: 사용자 실제 ID와 유호사용자 ID, 그룹ID 보여줌



- 특별한 실행파일을 실행할 때
- 유호 사용자 ID는 달라짐

Set -user -id 실행파일이 설정된 파일 실행하면

이 프로세스의 유호사용자 ID는 그 실행파일의 소유자 권한 갖게됨.
실행되는 동안 그 파일의 소유자 권한 가짐.

\$ ls -l /usr/bin/passwd

-rws r-xr-x 1 root root 27000 ...

set user id 실행권한이

설정된 파일이므로

password 명령어가

So LH PW로만 접근 가능

→ passwd, root 권한 접근 제어.

일반 사용자가 이 파일 실행하면 이 프로세스의

소유자는 root. 유호 사용자 ID는 root 대신 자신의 ID가 됨

Set - Group - id : 접근권한이 - r-xr-Sr-X
↓
접근부분에 X 대신에 들어감.

< Set - -id 설정 >

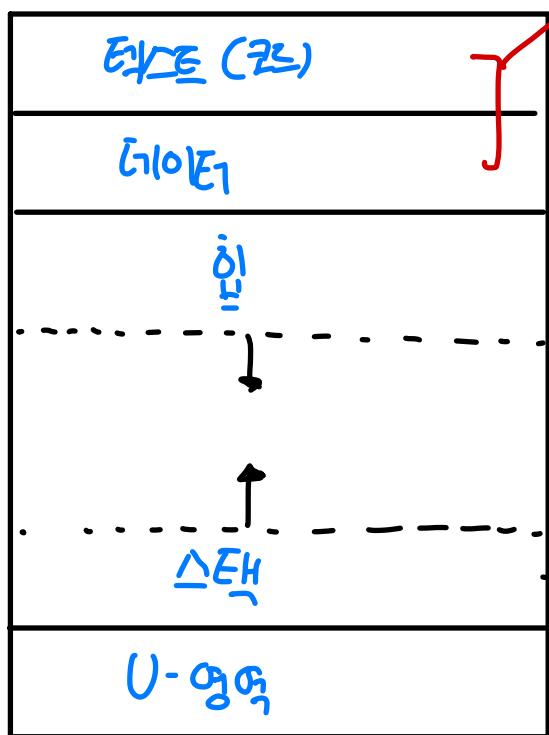
기준인 O . 46은 그룹.

Set - user - id : \$ Chmod 4755

Set - group - id : \$ Chmod 2755 group은 2.

프로세스 생성 (어려움) (과 1:13분)

Compile 시 풀



텍스트 (코드): 프로세스가 실행하는 실행 코드 저장 영역

데이터: 프로그램 내 전역 변수, 지역 변수 등을 위한 영역

→ 동적 메모리 할당을 위한 영역

(Runtime Stack)

→ 함수 호출을 구현하기 위한 실행시간 스택을 위한 영역

→ 열린 파일의 파일 디스크립터, 흐름, 작업 대책 등 프로세스의 내부 정보

\$ Size [실행파일]: 실행파일의 각 영역의 크기를 알 수 있음.
→ text, data 등 ...

→ 파일의 대상으로 찾.

07. 부팅

시스템 부팅 → 코어OS 실행 → 전/후면 처리 - · · ·

* 부팅 프로세스: test

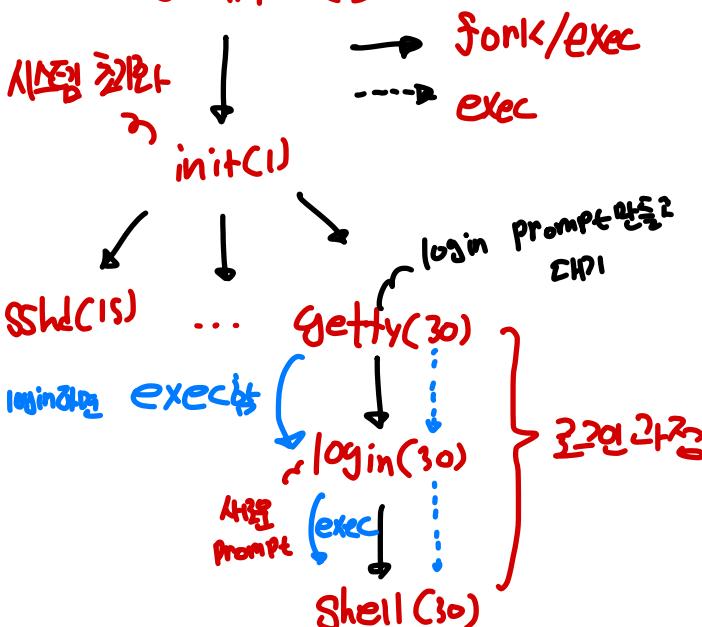
SSHd: Secure Shell

보안이 강화된 서비스 원칙

접속 및 제어 도구

* 시스템 부팅 : fork / exec 시스템 과정을 통해 이루어짐

Swapper (0) → Scheduler (코어OS 스케줄링 큐)



exec는 return 안 받고

자신이 차지한 시스템 리소스 둘러싸고

즉, '자기 대신' 됨 + 다른 프로세스 아니 보안 안정적임.

o getty 프로세스: login 프롬프트 내고 키보드 입력 대기

↓ exec

o login 프로세스: 사용자의 login ID, Password 검사

↳ 보안 강화 . why? login -shell 불가능 및 악성

↓ exec

o Shell 프로세스: 시작 파일을 실행 후 shell 프롬프트

사용자 명령어 대기

※ - 시스템 - root 부팅 프로세스

o Swapper (스케줄링 프로세스) PID = 0

커널 내부에서 만든 첫 번째 프로세스로
프로세스 스케줄링 큐.

o init (초기화 프로세스) PID = 1

etc/inittab 파일에 기술된 대로 초기화

서비스 제작은 디자이너가 계속 상수하는
서비스 대문 프로세스 프로세스

서비스들을 위한 대문 프로세스들이 생기면.

백그라운드 프로세스: 입출력 차단, 대화

터미널과의 관계를 끊은 모든 프로세스

디＃ 프로세스는 백그라운드 프로세스로

부모프로세스(PPID) 가 1 혹은 다른

디＃ 프로세스인 프로세스.

서비스와 요청에 대해 응답하는

요청 중인 실행중인 백그라운드 프로세스

↓

번호 fork하고 자신을 주어면

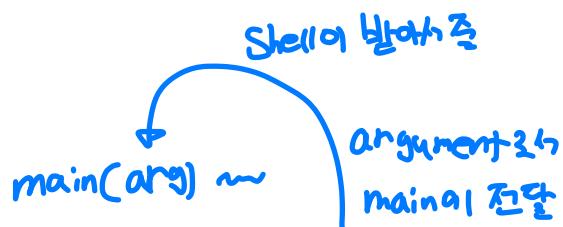
고려가-진 자식 프로세스를

init이 차지 밀었으니 다른凡是
방식

* 프로세스 확인

\$ ps tree: 실행중인 프로세스들의 부모, 자식 관계를 트리 형태로 출력

프로그램 실행



- exec 시스템 호출: [C 시작 루틴]에 명령줄 인수와 환경 변수 전달
프로그램을 실행시킴.

- C 시작 루틴 (Start-up routine)

main 함수 호출하면서 명령줄 인수, 환경 변수를 전달

exit(main(argc, argv));

실행이 끝나면 반환값을 넘아 exit 한다. // execute 모드

개념

exec 시스템 콜

exec(Parameter)로. 유저가 주입

사용자
프로세스

C 시작 루틴

호출

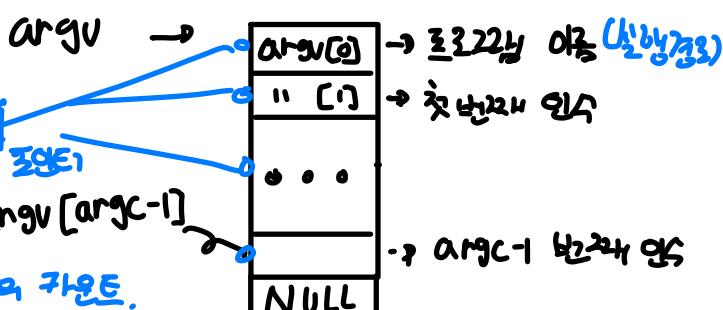
int main(int argc, Char * argv[]);

[명령줄 인수/환경 변수]

int main(int argc, Char * argv[]);

argc: 명령줄 인수의 수 (정도 개수)

argv[]: 명령줄 인수 리스트를 나타내는 포인터 배열 (main이 주는 실질적인)



4000

전면처리 / 후면 처리

foreground

전면처리: 직관적 - 입력된 명령어를 전면하게 실행

Shell은 명령어 끝날 때까지 대기

\$ 명령어

Shell 시작 \rightarrow A 프로그램 \rightarrow Shell : Shell 시작 끝나면 Shell로 옴

background

후면처리: 배그라운드 - 명령어를 후면하게 실행하고 전면처리는

다른 작업을 실행하여 동시에 여러 작업을 수행

\$ 명령어 & \rightarrow 이거 하나만 봐야면 background로 처리

Console 환경의 Shell은 prompt가 없지 않고

background의 형태로 Process를 두는

data dependency ($A \rightarrow B \rightarrow C$ 끝나고해야되는데, 의존성)에서는

전면처리는 가능하지만 후면처리는 불가능 X

• 후면작업 확인 : background 확인.

\$ jobs [%작업번호] : 후면에서 실행되고 있는 작업들을 리스트.

작업번호 명시하면 해당 작업만 리스트함.

후면 \rightarrow 전면 전환

\$ fg [%작업번호] : 작업번호에 해당하는 후면작업 \rightarrow 전면 작업으로 전환

\$ bg [%작업번호] : 작업번호에 해당하는 전면작업을 후면작업으로 전환하여 실행

입출력 재지정

Shell 장점: 작은 프로그램을 Function처럼 쓸: 한 줄의 재사용
or 32줄짜리 재사용



50

줄격 재지정: 방법 1, 의미 이해!

\$ 명령어 > 파일: 명령어의 훌륭한 출력을 다른 파일에 저장

ex) \$ who > names.txt : 이 시스템 유저를 명단 만들기

\$ Cat names.txt

Standard input으로부터 한 줄씩 읽음 (키보드)

\$ Cat <ⁱⁿ > list1.txt

Hi!

^P

• 줄격 추가

\$ 명령어 >> 파일: > 1개는 new file

>> 2개는 기존파일에 append (뒤에 추가)

\$ date >> list1.txt

\$ Cat list1.txt

Hi!

Thu Mar 12 18:45:26 KST 2020

입력 재지정

\$ 명령어 < 파일 : 명령어의 훈련입력을 키보드 대신 파일에서 받음.

• 문서 내 입력 (here document)

\$ 명령어 << 단어 : 명령어의 표준입력을 키보드 대신
...
단어 같은단어. 단어와 단어 사이의 입력을 내용으로 받는다.

오류자집

§ 명령어 2) 파일: 명령어의 풀준 오류를 모니터 대신에 파일이 저장.

표준 출력(Standard Output): 정상적인 실행의 출력

표준 오류 (Standard error): 오류 메시지 출력

```
$ ls -l /bin/usr 2> err.txt
```

\$ Cat err.txt

IS: Can not access /bin/usr : No such file or directory

파이프(Pipe)

＄ 명령어 1 | 명령어 2 : 명령어 1의 훈련 출력이 표시로 옮겨

\$ who | sort

시.운.나 악.글.나 악.과.별.소.으

정열
(목작우)

여러개의 명령어 사용하기

◦ 명령어 열 (Command Sequence)

\$ 명령어 1; ...; 명령어 n : 나열된 명령어를 순차적으로 실행

단축적 표현: 그룹에 속한 전체 명령어 같이 실행. 한꺼번에

◦ 명령어 그룹

\$ (명령어 1; ...; 명령어 n): 나열된 명령어를 하나의 그룹으로 묶어 순차적 실행

↳ (0%나온
조건 명령어 열)

\$ 명령어 1 && 명령어 2 : 명령어 1이 성공적으로 실행되면 2도 실행
아니면 X

ex) \$ gcc test.c && a.out
test.c 컴파일 → 성공하면 a.out 실행
성공하면 a.out 실행.(실패하면 error)

\$ 명령어 1 || 명령어 2 : 명령어 1실패 → 2실행
" 성공 → 2실행 X.

파일 이름 대치 - 명령어 대치

0 파일 이름 대치

① 대출문자를 이용한 파일 이름 대치: 대출문자 이용해 한 번의 대치로 여러 파일 나타냄

대출문자

예시

- * 번 스트링을 포함하여 임의의 스트링을 나타냄
- ? 임의의 한 문자를 나타냄
- [..] 대괄호 사이의 문자 중 하나를 대체하여 복수형의 사용 가능

\$ gcc *.c : C 확장자 대치

a?.c .c 는

\$ gcc a.c b.c zzz.c

a1c.c, a1kc.c ... 등 CL 규칙에

\$ ls *.c

but ac.c, acc.c 는 X.

\$ ls [ac]*

↳ a_onC 인 애들 전꺼는 예고하기
(예상)

0 명령어 대치

~의 ~의.

명령어를 대체할 때 다른 명령어의 실행 결과를 이용

• 명령어는 그 명령어의 실행 결과로 대체된 후에 실행

\$ echo 현재 시간은 `date`

현재 시간은 Thu Mar 12 18:08:25 KST 2020

echo: 주석 문자열을

\$ echo 현재 스实践中 내의 파일 개수 : `ls | wc -w`

개별문자 갯수하여

현재 디렉토리 내의 파일 개수 : 32

word Count
-w는 단어

파일을 찾는 명령어

파일을 찾는 명령어

시그널

(중단, 사인트: interrupt)

- 악기지 않은 사건 발생 시 이를 알리는 SW 인터럽트

시그널 발생 예

SIGFPE 낮은 산술 오류

주요 시그널 PPT에 보기.

SIGPWR 정전

악기 + 기본적으로 종료인지 무시인지

SIGALRM 알람시계 유틸

→ 고어링트: 잘못된 명령어, 심각한 산술 오류
(0으로 나누기) 등 문제기가 생기면
사태를 안 볼 속자로 만드는가.

SIGCHLD 자식 프로세스 종료

SIGINT 키보드 → 종료 요청 (Ctrl-C)

SIGSTP 키보드 → 정지 요청 (Ctrl-Z)

시그널

Kill 명령어: 한 프로세스가 다른

프로세스를 제어하기 위해 특별 프로세스
임무를 시그널을 강제로 보내기

\$ kill -1

1) SIGHUP 2) SIGINT..
:

\$ kill [-시그널] PID

\$ kill [-시그널] %작업번호

주요 시그널

시그널 이름	의미	기본 처리
SIGABRT	abort()에서 발생되는 종료 시그널	종료(코어 덤프)
SIGALRM	자명종 시계 alarm() 울림 때 발생하는 알람 시그널	종료
SIGCHLD	프로세스의 종료 혹은 중지를 부모에게 알리는 시그널	무시
SIGCONT	중지된 프로세스를 계속시키는 시그널	무시
SIGFPE	0으로 나누기와 같은 심각한 산술 오류	종료(코어 덤프)
SIGHUP	연결 끊김	종료
SIGILL	잘못된 하드웨어 명령어 수행	종료(코어 덤프)
SIGIO	비동기화 I/O 이벤트 알림	종료
SIGINT	터미널에서 Ctrl-C 할 때 발생하는 인터럽트 시그널	종료
SIGKILL	잡을 수 없는 프로세스 종료시키는 시그널	종료
SIGPIPE	파이프에 쓰려는데 리더가 없을 때	종료

시그널

PID(작업번호)로 지정된 프로세스(

운영하는 시그널을 보냄.

DOSIX은 SIGTERM 시그널을 보내

해당 프로세스 강제종료

0 컴파일 기본

• GCC 컴파일러

\$ GCC [-옵션] 파일

C 프로그램을 컴파일 함.

유선 사용 안하고 실행파일 a.out 생성.

ex)

\$ gcc hello.c

\$./a.out

Hello World!

Object 파일 저장

\$ gcc -o hello hello.c

\$ hello

Hello World!

(안나온다)

0 자동빌드 도구

다른사람의 소스코드 재활용

(SPK, Library ...)

여러코드쓰고 복잡해짐(버그, 기능오류)

다중 모듈 프로그램을 구성하는

의존성

일부파일이 번경된 경우

컴파일 시간↑

번경된 파일만 컴파일하고, 파일들의

복잡도↓

의존 관계에 따라 필요한 파일만 다시 컴파일하여 실행파일을 만듬.

↳ make 시스템: 대규모 프로그램의 경우 헤더, 소스파일, 실행파일, 빌드 파일의 모든 관계를 기억하고 관리하는 것이 필요



make 시스템 이용. 효과적인 작업

0 종료시킬 때하기 SIGKILL

\$ kill -9 { 프로세스번호

\$ kill -KILL 프로세스번호

0 다른 사람을 보내기

\$ 명령어

[1] 1234

\$ kill -STOP 1234

\$ kill -CONT 1234

\$ make [-f makefile] : Make 시스템은 makefile을 이용하여 빌드 실행도구를 빌드
혹은 사용하여 빌드의 메모리 파일 저장 가능

o Makefile 파일

실행 파일을 만들기 위해 필요한 파일 리스트로
그들 사이의 의존 관계
만드는 방식을 기술

o make 시스템

Makefile 파일은 파일의 상호 의존 관계를 파악,
실행 파일을 '쉽게' 다시 만듦.

Makefile 파일 구성

목표 (target) : 의존 리스트 (dependencies)

명령리스트 (Commands)

(예시)

O: object, C: C구문

main : main.o test.o

gcc -o main main.o test.o

main.o : main.c

gcc -c main.c

test.o : test.c

gcc -c test.c

전역 혹은 Local 가
내부 같아도 고려한가 취급.

o make 실행

\$ make 혹은 \$ make main

gcc -c main.c
gcc -c test.c }
gcc -o main main.o test.o

바깥에 있으면 make 해도 안만듬.

is test.c 파일 수정한 후에는

\$ make ~> main.c 다시 만듬.

gcc -c test.c

gcc -o main main.o test.o

Process 생성

복습

- 프로세스가 하드웨어 인터럽트에 의해 CPU를 빼앗기게 되면 Preemptive scheduling이라고 한다.(time interrupt) -> 자기 의지와 상관없이 CPU를 빼앗기는 스케줄링
- 반면 Non-preemptive scheduling은 소프트웨어 인터럽트에 의해 일어난다. -> 내가 I/O를 기다리는데 I/O가 끝나면 오래 걸릴 것 같다. 결국 내가 지금 CPU를 점유하고 있는 게 별 의미가 없다. 이럴 경우 OS에게 소프트웨어 인터럽트를 걸어서 CPU를 넘기는 것
- 그리고 인터럽트와 인터럽트 서비스 루틴을 이용하여 Context switching이 일어난다.

일반적인 프로세스 생성 방법

프로세스가 생성이 되려면 OS가 그 프로세스의 Context들을 다 만들어 줘야 한다.(ex:Fake stack 생성 등..)

1. 먼저 프로세스가 생성이 되려면 파일 시스템의 그 대상이 되는 exe파일이 있어야 한다. 이 exe파일의 Path를 OS에게 전달 해야한다.
2. OS는 실행 가능한 파일들의 코드를 읽어 들인다. -> 즉 Memory Context의 한 부분인 Code 세그먼트로 exe 파일의 코드들을 읽어 들인다.
3. 또 exe파일에는 전역 변수의 선언과 정보 들이 있다. 이 정보들을 기반으로 Data 세그먼트에 알맞게 영역을 잡아준다.
4. 이렇게 Code 세그먼트와, Data 세그먼트로 exe파일들을 불러들이는 작업을 프로그램을 로드 한다고 말한다.
5. 스택 세그먼트와, 힙 세그먼트는 아직 아무것도 없으므로 초기화만 시킨다.
6. 그리고 그 프로세스의 정보를 담은 PCB(Process control block)를 Malloc 해서 필요한 정보를 채워 넣는다.
7. 이렇게 생성된 PCB를 레디큐에 장착시킨다.

UNIX 계열의 프로세스 생성

- 그런데 유닉스 계열의 OS는 시스템이 부팅 할 때 0번 프로세스만 위와 같은 방법으로 생성하고 나머지 프로세스는 다른 방법으로 생성한다. -> 복제라는 특별한 기법을 사용한다.
- 복제하는 역할을 하는 System call 이 fork() 이다.
- 유닉스에서는 어떤 프로세스가 생성이 되려면 그 프로세스를 생성하라고 명령을 내려주는 다른 프로세스가 있어야한다. 나를 만들어주는 프로세스를 부모 프로세스라고 하고 그리고 만들어진 프로세스를 차일드 프로세스라고 한다.
- Parent Process : Process Cloning을 초래하는 기존의 Process
- Child Process : Parent Process로부터 만들어지는 새로운 Process

fork() 시스템 콜

- Parent Process가 Child Process를 만들기 위해서는 fork()라는 시스템 콜을 호출한다.
- fork()라는 시스템 콜이 호출되면 OS가 fork()를 호출한 부모프로세스를 중단시키고 부모프로세스가 가지고 있던 context들을 사진을 찍어서 그 사진 찍은 context를 그대로 복사한다. 이런 과정으로 자식 프로세스를 생성한다.
- 그렇지만 프로세스 ID는 다른 값을 갖게 된다. But 프로세스 ID를 제외한 모든 값을 그대로 복제하기 때문에 부모 프로세스, 자식 프로세스는 똑같은 형태로 생성된다.
- 그리고 나서 자식 프로세스의 PCB를 레디 큐로 보내고 나면 fork() 시스템 콜의 수행은 끝나고 다시 부모 프로세스로 돌아간다.

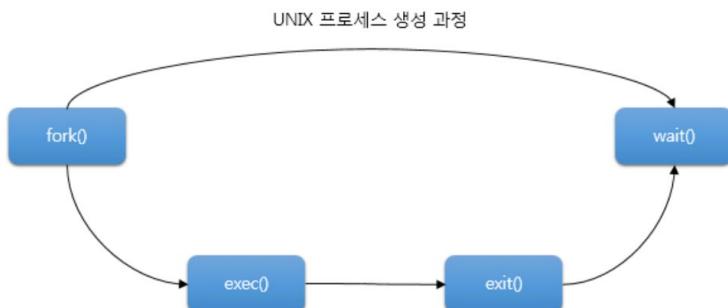
for() 시스템 콜이 가지는 문제점

- 매번 모든 Context의 복사본을 만드는 것은 매우 비효율적이다.
- 컴퓨터 시스템은 맨 처음 만든 프로그램 프로세스(0) 이외에는 다른 프로그램을 동작 할 수 없다. 말이 안 된다. -> 계속 복사되기 때문에...
- 그래서 UNIX OS는 fork()를 한 다음 반드시 exec()라는 시스템 콜을 호출한다.

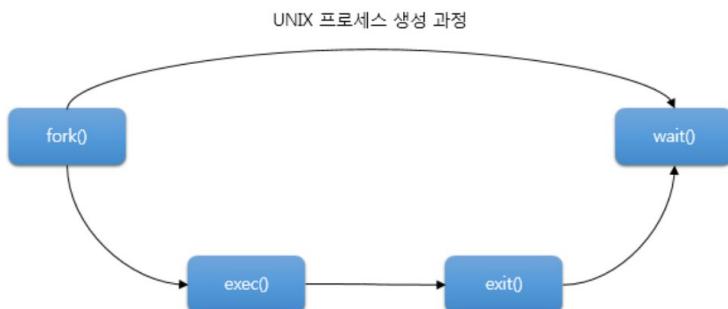
exec()

- exec()는 fork()가 가지는 문제점을 해결하기 위해 만들어진 시스템 콜이다.
- exec() 파라미터는 바로 생성될 프로그램의 exe파일의 path name 들이다.
- 생성될 exe 파일의 코드와 데이터들을 이미 만들어진 프로세스(fork())를 이용해 부모 프로세스를 복사해서 가지고 있는 자식 프로세스(에 딛어 씌워버린다(오버라이드).
- 이 동작을 완료하면 새로운 프로그램으로 출발 시킬 수 있다.

UNIX 계열의 프로세스 생성 과정



UNIX 계열의 프로세스 생성 과정



- 부모 프로세스가 fork()를 한다. fork()를 하는 시점에 자식 프로세스는 부모 프로세스의 context를 그대로 복사하고 exec()을 수행한다.
- exec()을 수행함으로 부모 프로세스와 자식 프로세스가 각각 다른 프로그램을 수행 할 수 있도록 된다.
- 부모 프로세스는 fork()라는 시스템 콜을 하고 wait() 시스템 콜을 한다. 이 wait() 시스템 콜의 argument(인자)는 자기가 생성한 자식 프로세스의 ID이다.
- wait()는 그 매개변수로 준 프로세스가 수행을 종료 할 때 까지 나를 기다리게 하라는 것
- 자식 프로세스는 exec()을 해서 코드를 새로 얻어 수행을 하고 다 수행하면 exit()라는 시스템 콜을 호출한다. 이 exit() 시스템 콜은 프로세스를 종료시키는 시스템 콜이다.
- 우리는 지금까지 C를 짜면서 exit()를 써본 적이 없다? -> OS가 자동적으로 exit()를 호출해주도록 하여 프로그램을 종료 시키도록 하게 한다.
- 즉 exit()는 그 프로세스가 가지고 있던 데이터구조, 자원을 다 가져가는 것을 말한다. 하지만 exit()는 종료가 잘 되었나 안 되었나를 확인하는 코드 하나만을 남겨 놓고 이 코드를 wait()에 전달한다.
- 예전에 유닉스에 좀비 state란 게 있다고 했는데 이 좀비state가 바로 exit()를 마친 프로세스를 말한다. -> 모든 것을 다 가져가는데 딱하나 Exit Status만 남겨진 상태

Zombi state

exit()을 마치고 Parent Process가 자신의 Exit Status를 읽어가기를 기다리는 Process 상태

Shell

- 요즘은 그래픽 유저 인터페이스가 발전되어 마우스로 OS와 의사소통을 하지

Shell

- 요즘은 그래픽 유저 인터페이스가 발전되어 마우스로 OS와 의사소통을 하지 만 과거에는 텍스트 기반이었다.
- Shell Command line Interpreter를 사용하여 OS와 의사소통을 했다. -> OS가 유저에게 명령어를 입력 하세요 라고 말한다. 유저는 명령어를 입력해 Return 키를 치면 명령이 들어가게 된다. OS는 그 명령에 따라 프로세스를 생성시키고 다시 명령을 넣으라고 나왔다. -> 명령어를 넣으세요(Prompt): 리눅스의 경우 프롬프트를 \$ 으로 표시한다.
- Shell 이 어떻게 구성되는지 아래 코드가 보여주고 있다.

```
for(;;)
{
    cmd = readcmd();
    pid = fork();

    if(pid < 0)
    {
        perror("fork failed");
        exit(-1);
    }

    else if(pid ==0)
    {
        // fork() 후 child 프로세스에서 exec() 수행
        if(exec(cmd) < 0) perror("exec failed");
        exit(-1);
    }
    else
    {
        // 부모 프로세스는 자식 프로세스가 exit status 를 넘길때까지 대기
        wait(pid)
    }
}
```

1. readcmd() 함수는 사용자에게 프롬프트를 보내주고 명령어를 넣으라고 말 해주는 함수이다. 사용자가 리턴 키를 치면 입력한 라인버퍼를 cmd 에 저장 시킨다. -> cmd에는 유저가 실행시키고자하는 exe파일의 path name들이 저장될 것이다.
 - 그 다음 이 Shell은 저장된 커맨드를 실행시킬 차일드 프로세스를 만든다. -> fork() System call
2. fork() 시스템 콜을 호출하면 리턴 값이 있다.(음수면 에러, 0이상이면 정상 동작)

3. 만약 성공하면 pid가 0인지 그렇지 않은지 검사 한다. 여기서 pid는 fork() 시스템 콜에 의해 생성된 자식 프로세스의 ID를 말한다.
4. 그런데 자식 프로세스의 ID가 0인 것이 좀 이상하다. 이것이 가능한 일일까? (ID가 0인 것은 유닉스가 맨 처음 부팅할 때 만든 프로세스 즉 부모 프로세스이고 나머지는 다 복제하므로 0인 프로세스는 부모 프로세스일 텐데?.. 즉 자식 프로세스의 ID로 0을 줄수 없다.) 이 루프를 들어오는 것은 불가능하지 않을까? 일단 이 부분은 무시하고 아래에서 설명하겠다.
5. pid가 0이 아니면 마지막 else 문에 가서 차일드 프로세스가 끝날 때 까지 부모 프로세스는 wait()를 한다. (콘솔창에서 명령을 입력하면 수행이 종료될 때 까지 유저는 타이핑을 못한다. 즉 프롬프트를 못하는데 그런 동작이 이 부분이다.)
6. pid가 0이 되는 부분은 바로 자식 프로세스가 실행되는 부분이다. -> fork()를 호출한 시점에서 부모프로세스 수행이 딱 중단되고 Context들이 사진 찍히듯이 찍힌다. 그리고 나서 그 모든 내용이 자식 프로세스로 복사가 된다. -> 자식 프로세스의 PCB가 생성이 되고 그 PCB가 레디 큐에 들어가게 된다. -> 결국 자식 프로세스의 PCB가 선택되어서 스케줄링을 받아서 수행된다.
7. 그러면 자식 프로세스는 부모 프로세스의 코드를 그대로 복사했기 때문에 동일한 코드가 수행될 것이다. 과연 이 자식 프로세스는 복사된 부모 프로세스의 코드 중 어느 부분부터 수행이 될까? -> ***바로 fork() 다음부터 수행될 것이다. 왜냐하면 시스템 콜 호출되면 SW 인터럽트가 발생되고 그리고 리턴 어드레스가 스택에 들어가기 때문이다(리턴 어드레스는 바로 fork() 다음으로 지정 되어있다.) ***
8. 그 결과 자식 프로세스도 한번도 fork()를 호출한 적이 없지만 스택의 리턴 어드레스로 fork() 다음 위치가 기억되 있기 때문에 그 위치로 가게 된다.
9. OS는 여기서 한 가지 트릭을 더 사용 한다. -> 자식 프로세스에게 리턴 할 때는 리턴 값을 0으로 설정한다.
10. 그렇다면 이 리턴 값을 어떻게 넘길까? -> CPU 레지스터를 통해 넘기는데 CPU 레지스터 값이 스택에 저장이 되어있기 때문에, 리턴 값을 담고 있는 CPU레지스터가 저장된 스택 필드에 부모 프로세스인 경우는 자식의 pid를 넣고 자식 프로세스인 경우는 0을 넣는다.
11. 그래서 자식 프로세스는 fork()를 한번도 수행한적 없지만 fork() 다음을 수행하고 pid가 0이므로 그 분기문안의 명령을 수행한다. -> exec() 시스템 콜 수행
12. 그리고 나서 자식 프로세스의 수행이 종료가 되면, exit()이 호출되고 exit()

12. 그리고 나서 자식 프로세스의 수행이 종료가 되면, exit()이 호출되고 exit()에 의해 부모로 시그널이 넘어가서 부모 프로세스가 깨어나게 되고 다시 프롬프트가 실행된다.

정리

- 실제 Child Process의 Pid는 0이 아니고 다른 값이다. Child Process에게 전달되는 fork() 시스템 콜의 리턴값만 0인 것이다. -> 부모 프로세스한테 넘길때는 실제Pid 값, 자식 한테 넘길때는 0으로
- 위 내용으로 알 수 있듯이 리턴된 값 0은 프로세스의 id를 의미하는 것이 아니라 단지 리턴이 성공적으로 되었음을 알리기 위함이다.

왜 fork() 와 exec()를 사용할까?



- CPU가 실제로 발생시키는 주소와 메모리의 Target address가 실제적으론 같지 않다.
- 그 사이에 이 주소를 변환해주는 장치들이 있다(MMU) 이 MMU에 입력되는 주소를 logical address(논리주소) 그리고 실제 메모리로 가는 주소를 physical address(물리주소)
- 즉 물리주소는 32bit CPU에 256MB를 꼽았다면 0~256MB 번지까지가 물리주소가 된다.
- 논리 주소는 뭘까? -> 실제로 존재하지 않지만 그 프로그램이 간주하는 가상적인 메모리이다.
- 32bit 머신이 있다고 가정하면 -> 주소공간은 0 ~ $2^{32}-1$ 까지 주소공간을 만들 수 있다. 이것이 바로 논리주소공간이다.
- 이 논리주소공간은 모든 프로세스가 한 개씩 가지고 있다.(MMU로 구현함)
- 결국 모든 프로세스는 독자적인 논리주소공간을 가진다.

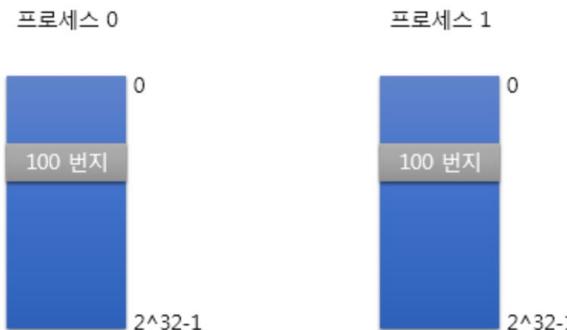
두 프로세스는 어떻게 통신을 할까?

프로세스 0

프로세스 1



두 프로세스는 어떻게 통신을 할까?



- MMU가 위 논리주소를 물리주소로 매핑 해준다는 것을 알았다.
- 그럼 process 0 의 100번지와 process1의 100번지가 같은 주소일까? 다른 주소일까? -> 다른 주소이다.
- 논리주소 공간이 다르면 주소번지가 같다고 하더라도 다른 주소가 된다.
- 그런데 이것이 문제가 되는 것이 proc0 과 proc1 이 의사소통을 해야 하는데 할수가 없으므로 별도의 추가적인 방법이 필요하다
- 결과적으로 파일을 이용해 서로 의사소통하는데 여기서 fork() 가 사용된다.
- 아래 소스가 fork() 가 필요한 이유이다.

```
int fd;           // 파일 시스템변수

foo()
{
    fd = open("pile");           // 파일을 집어 넣는다

    if(fork() == 0)             // 자식일 경우
    {
        read(fd,...);          // 하나의 파일로 서로 의사소통함
    }

    else                       // 부모일 경우
    {
        write(fd,...);         // 하나의 파일로 서로 의사소통함
    }
}
```

```

int fd;           // 파일 시스템 변수

foo()
{
    fd = open("file");      // 파일을 집어 넣는다

    if(fork() == 0)          // 자식일 경우
    {
        read(fd,...);       // 하나의 파일로 서로 의사소통함
    }

    else                     // 부모일 경우
    {
        write(fd,...);      // 하나의 파일로 서로 의사소통함
    }
}

```

COW(Copy On Write)

- 부모프로세스를 카피했는데 exec()으로 한번 더 오버라이트 하는 불필요한 문제(성능저하)들을 해결하기 위해 개발자들이 특별한 메커니즘을 만들어냄
-> COW
- Process Context를 fork() 시점에 복사하지 않고, Data Segment에 새 값이 쓰여질 때 복사하는 기법
- 즉 부모가 fork()해서 자신의 context를 카피 할때는 실제적으로 내용을 복사하지 않고 포인터 자료구조만 만들어서 자기의 code 세그먼트, Data 세그먼트 를 자식 프로세스가 가리키게만 한다. stack, heap 은 별도로 만들어야 한다.
- 그리고 exec() 할 때 비로소 복사가 된다.

프로세스 제거

- exit() 호출
- abort() 호출 : 부모가 신호를 보내서 프로세스를 죽임

[OS]프로세스 생성과 제거 was published on February 09, 2016 and last modified on February 13, 2016.

08. 인터넷 딱 3문제 나눠, 어느정도 알려준다.

Inter + net] TCP/IP : Internet Protocol
Interconnected Network] 상호연결

Internet Protocol] 주약

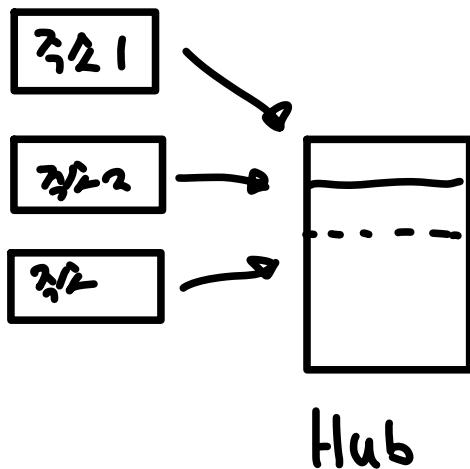
Internet Address : 32 bit IP v4

$$0 \sim 255 \quad 2^8, 8\text{bit} \times 4 \Rightarrow 192.168.0.1$$

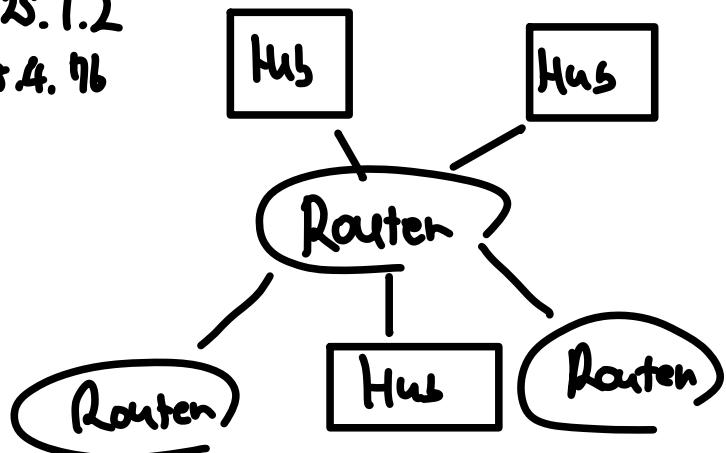
NAT 가상주소

卷之三

22bit 2 전시회 연출



Part 2 - 198.125.1.2
Part 2 - 221.128.4.96
:
:



Node.js 주제설명.

Router Table

주소록 Routen 앱기 이 주소

가족은 어떤데 전달.

while ~~fund~~.

A node to B node 정보로 쌍가능

IP 주소 \Rightarrow Node 찾을 수 있음.

이름, 명칭



도메인 이름 도메인 주소 \rightarrow IP 주소 Change

ex) google.com 의 실제 주소: 172.217.25.10 이다 가능하면

Domain Name Service : DNS

도메인 이름 물어보면 (사이트 Enter)



DNS Server 가 주소 중

도메인 주소 \rightarrow DNS \rightarrow IP 주소 \rightarrow Hub \rightarrow Router \rightarrow Router ...
 \rightarrow get http://www.
Packet Switch

\rightarrow Google Server

HTTP 요청을 받는다

인터넷 = 네트워크의 연결

네트워크 구조

LAN: Local Area Network

근거리 통신망.

구현 방식: 이더넷 (Ethernet)

일반적으로 사용

Router: 2개 이상의 네트워크 연결 장치 (로우터, 스위치, 라우터)

data Packet 흐름이 측정하기 편하게 따라 다른 대로 보낸다.

- 게이트웨이: 네트워크의 고용량 라우터. LAN을 인터넷에 연결하는 쪽에 장착됨.
- 인터넷: TCP/IP 2개의 종류 프로토콜을 이용해 정보 주고받는 공개 컴퓨터 통신망
- 프로토콜: 상호 호환성
 서로 다른 기종의 컴퓨터 사이에 어떤 자료, 어떤 방식, 언제 주고 받을지 등을 정해 놓은 규약 (국제규약 ISO)

OSI 7 계층 & TCP/IP 4 계층: 서비스, 통신 중요 but 유익 X
So PDF 봐.

TCP/IP

TCP/IP 프로토콜

TCP (Transport Control Protocol) in-order

- IP 위에서 동작하는 프로토콜 1, 2, 3, *, 5, 7, 6으로 오면
- 데이터의 전달을 보증하고 보낸 순서대로 받기 대비 1, 2, 3, 4, 5, 6, 7로 넘기 수준

IP (Internet Protocol)

- 호스트의 주소지정과 대기 보관 및 조립 기능이 대안 규약
- 인터넷 상의 각 컴퓨터는 자신의 IP 주소를 가짐.
- IP 주소는 네트워크마다, 장치마다 서로 인식, 통신하기 위한 주소

예: 203.252.201.11

IP 주소(IP Address)

- 컴퓨터 네트워크에서 전 세계 컴퓨터에 부여된 고유식별 주소

· 이진으로 B는 컴퓨터 중복 X

· IPv4는 32bit의 고갈위험 → IPv6로 대체될 줄 알았는데

IPv6가 자연적이다

NAT(Network Address Translation)
의 보호, 사용 IP 사용으로 필요로 줄어듬.

- 공인(Public) IP 주소

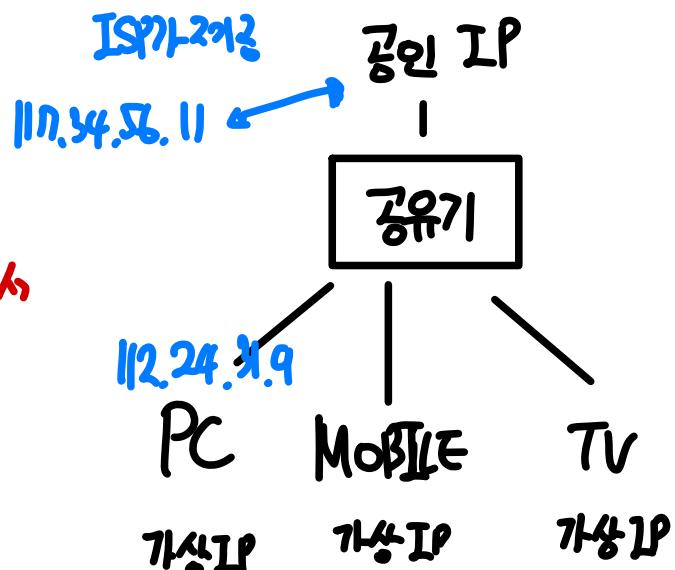
공인기관에서 인증한 공개형 주소로

외부에 공개되어 있어 다른 컴퓨터에서,

정식, 전용 기관은 IP 주소

- 공개되어 있어서 보안을 위해

방화벽 등이 필수

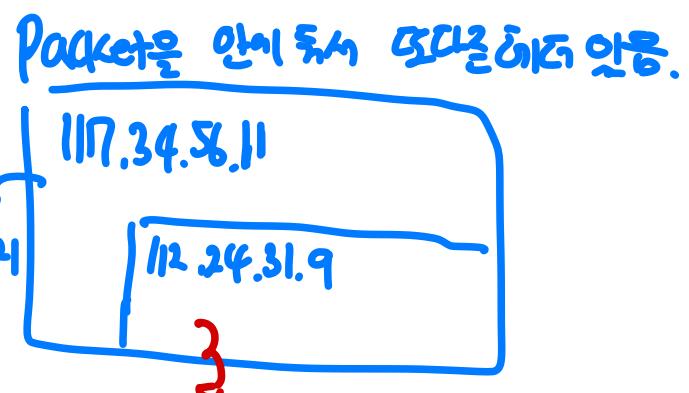


- 사내(SPrivate) IP 주소

가상 IP 주소와 결합하여 공연하지 않은
가상 IP 주소이기 때문에 외부에서,
쓰는 주소

정식, 전용 불가능

- 공유기: 하나의 공인 IP를 대체하여
기기가 공유해서 사용할 수 있도록
가상 IP 할당



공인 IP보다 Packet 풀고

내부의 공유기의 해당

Packet(가상IP, 사내IP) 등.

호스트명과 IP주소

• 인터넷이 연결된 컴퓨터에게 부여되는 고유한 이름

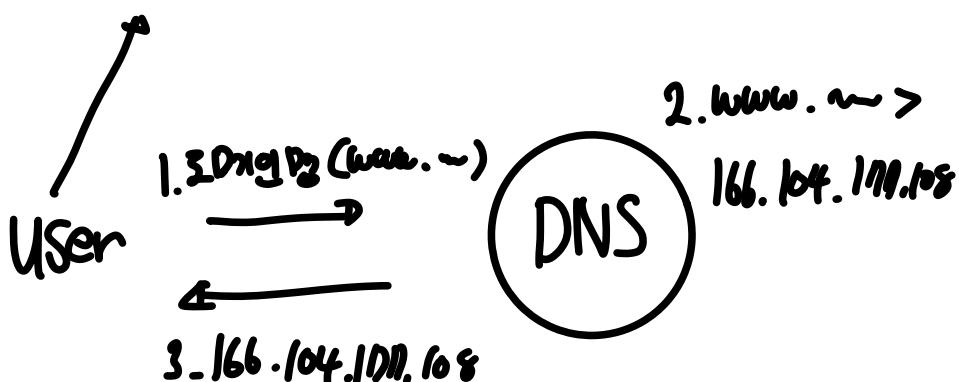
. 호스트명은 네트워크에 있는 모든 가능

. 도메인 이름 (Domain Name)라고도 함

ex) Domain IP
www.hanyang.ac.kr 116.104.177.108

DNS

4. Router ~ www. ~ 접속!



\$ hostname: 사용중인 시스템의 호스트명 출력

\$ ip addr: IP주소 출력

\$ nslookup 호스트명: 해당하는 호스트의 IP주소 알려줌

World Wide Web, WWW, W3

- 인트루넷이 연결된 컴퓨터들을 통해 사람들이 정보를 공유할 수 있는 전기자적인 정보공간

정보를 보면 link 개념.
Graph

하이퍼텍스트 (hyper-text)

- 문서 내의 어떤 위치에서 하이퍼링크를 통하여 연결된 문서나 이미지가 쉽게 접근

HTML (Hyper Text Markup Language)

하이퍼, 텍스트, 작성, 언어

HTTP (Hyper Text Transfer Protocol)

https://www.google.com/
www로 접속할 https로 할

Web Server ←→ Client
통신 시 사용하는 Protocol

한

URL (Uniform Resource Locator) : 웹접근주소

인터넷상의 존재하는 여러가지 자원들이 대체로 주소처럼

Web browser : 웹브라우저, 정보검색하는 데 사용하는 SW

Mosaic → NetScape → Internet Explore → ... Chrome

09. 소켓 : 사용자 통신

\$ write 사용자명 [단말기 명]

현재 로그인 되어 있는 다른 사용자에게 메시지를 보냄.

write all

\$ wall [화일]

현재 로그인 되어 있는 모든 사용자에게 메시지를 보냄.

화일 내용을 메시지로 보낼 수 있음.

시작 X

FTP (File Transfer Protocol)

\$ ftp -n [호스트 명]

\$ Sftp -n [호스트 명]: Secure ftp

{

FTP mget, mput

Multiple get / set.

파일 전송은 위한
FTP 서버 ←→ Client
주로 파일 업로드, 다운로드
ftp.hanyang.ac.kr

• 원격접속

☞ local의 Shell 가려 같다.

서로다.

\$telnet [호스트명 (혹은 IP주소)]

지정된 원격 호스트에 원격으로 접속한다.

• 안전한 원격접속 (Secure Shell. 터미널 이용 가능하지 않은) (사용자 관리)

\$ ssh 사용자명 @호스트명] telnet + 강력한 보안성

\$ ssh -l 사용자명 호스트명

SSH → 터미널 → 데스크톱인가? Ping은 네트워크 문제인지 아닌지

\$ Ping 호스트명 : 지정된 원격 호스트가 되는가 가능하지 기록화가 상황 확인 강제적 치료

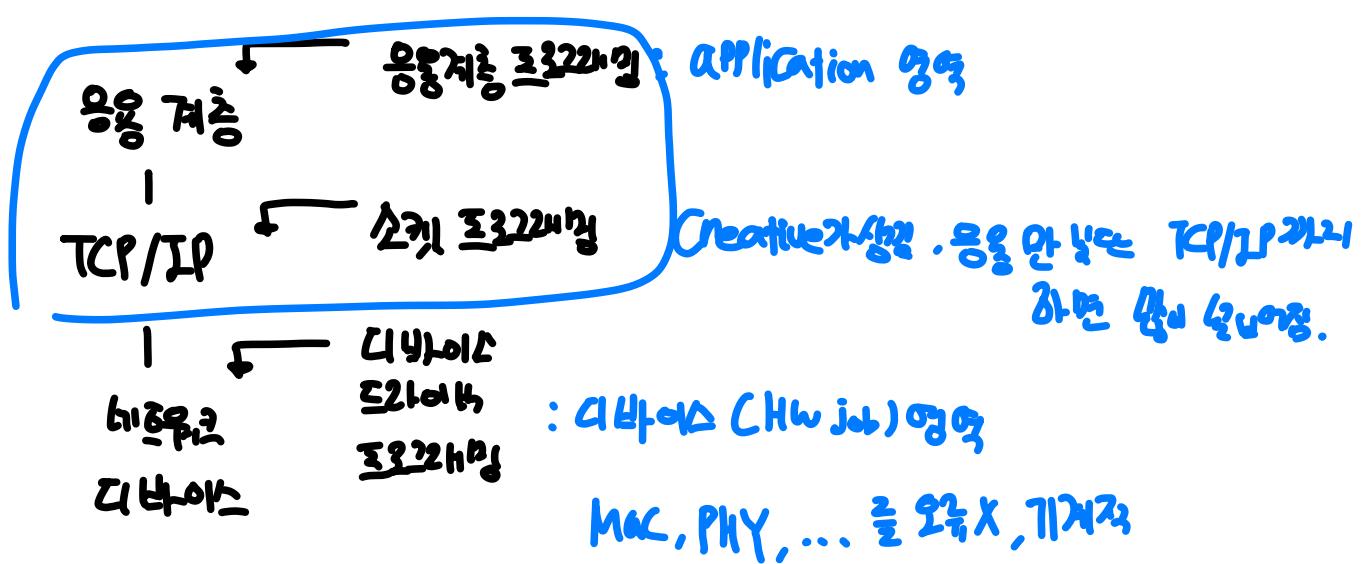
• 원격 디스플레이 예제

☞ Tip : 학생 때 Shell을 많이 써보니 일어나는 Shell, 그리고 GUI

PDF 보기

09. 소켓에서도 간단히 문제 날기. 아마 소켓도 일상적인 예제 안내용

네트워크 프로그래밍 제1장 복습 (간략)



Application 영역: PDF에 내용 많지만 일단 네트워크 수신 (기술적 장벽)
인터넷은 적용하기 어렵다.

반대로 이미 네트워크 상에서 데이터가 있는 경우 간단히 쓸 수 있다.

소켓 프로그래밍

TCP, UDP 같은 소켓(트랜스포트 계층)의 기능을 직접 이용

데이터 그램 단위의 Data 송수신 처리

대표적인 API (Application Program Interface): UNIX BSD Socket
Winsock

디바이스 드라이버 계층 프로그래밍 (MAC, PHY, ...)

- OSI 계층 2 단계의 인터페이스를 직접 다루는 프로그래밍
- 프레임을 솔루션별 기능만을 이용
 - ↳ 흐름제어, 오류제어, 인트, 네트워크 관리 등 ...
↳ Creative는 것 안됨. 표준적 예.

클라이언트 - 서버, 모듈 (일반적으로 서비스로 적어 놓기)

Client: 서비스 이용 장벽

Server: 서비스 제공 장벽

PDF. 다수의 네트워크
3Tier 구조.

소켓의 정의

TCP나 UDP 같은 트랜스포트 기능을 이용한 API

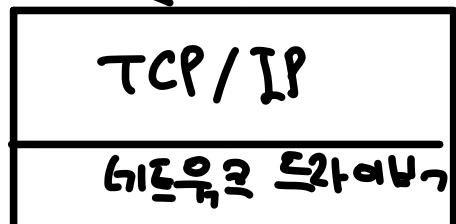
소켓: 네트워크에 대한 사용자 API

인터넷 프로토콜

- 양방향 통신 방법임. 클라 - 서버 모델 기반
하고 있어 프로세스 사이 통신이 매우 적합

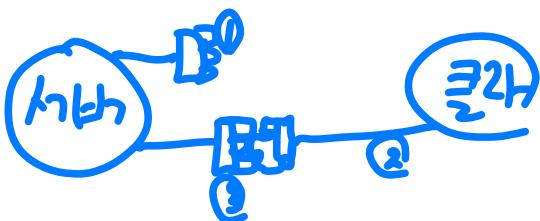
우리스 소켓 ≡ 인터넷 소켓. 비슷함.

응용 1 2 3
| | |
소켓 1 2 3



소켓 연결 (Connection*)

1. 서버가 소켓 만들기
2. 클라가 소켓 만들고 서버의 연결 요청
3. 요청 수락, 연결



스켓 연결 과정 (TCP/IP 네트워크 기반 이모습임.)

클라이언트

Socket

Connect

연결 요청



서버스 요청



응답 처리

4



EOF

Close

서버

Socket

bind

listen

wait

Accept

fork : PID가 다른 child

부모는 자식 PID

서비스 요청 처리



서비스 응답



Close

다음 클라이언트

연결 요청을 기다림

인터넷 소켓으로 표준 네트워크

{
Port, 봇구, 주소 서비스는
고유 포트 번호 얻음
(국제 표준화 됨)
:
:

.인터넷 소켓 -

인터넷 상호호스트 : 32bit IP주소 (42억개 공간) , domain Name

DNS . 호스트 엔트리 구조체

잘 알려진 서비스의 포트 번호

기인기 자신의 세포로로로 사용금지.

Web 서버 - 80번포트

10000대 이상 권장.

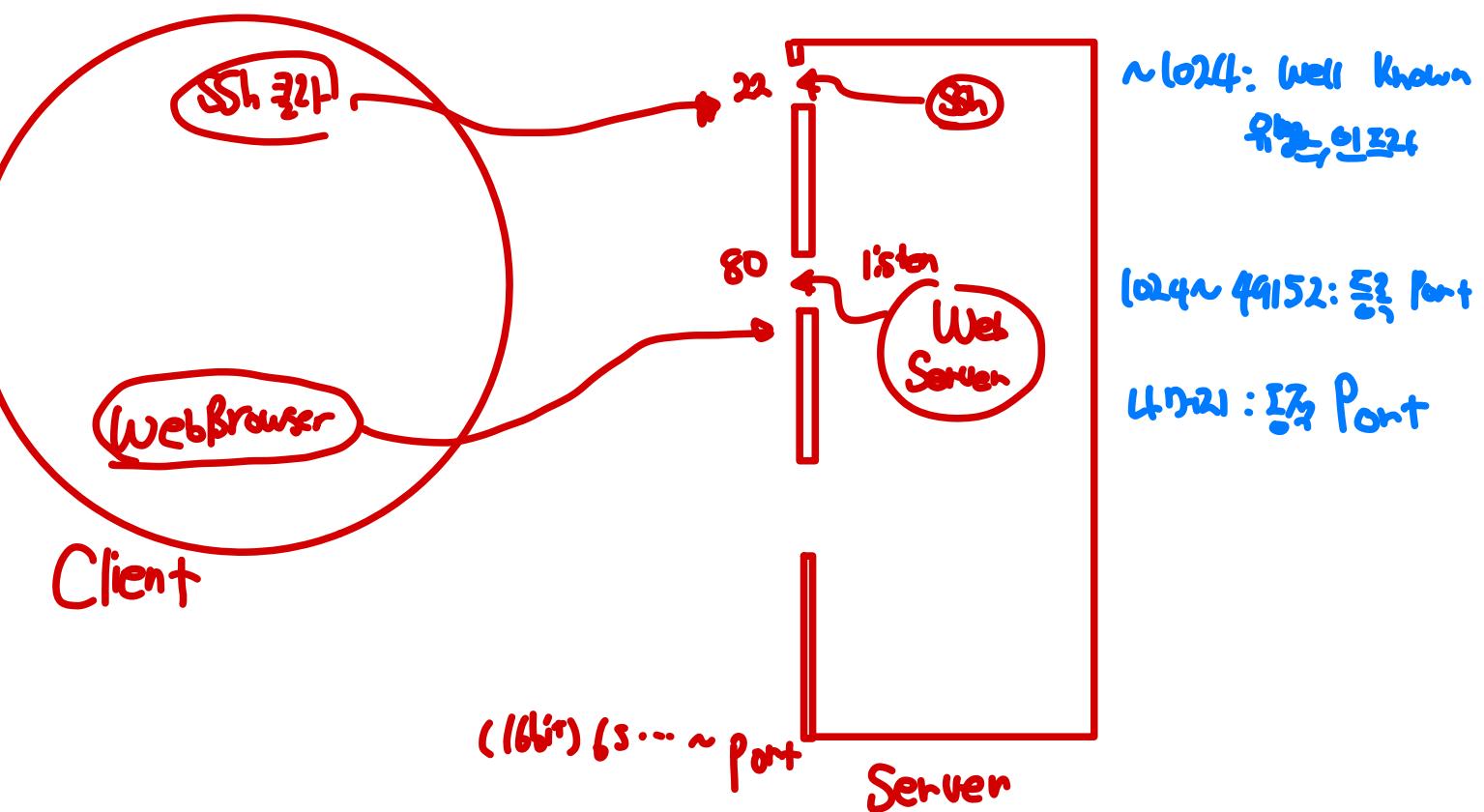
time 서버 - 13번포트

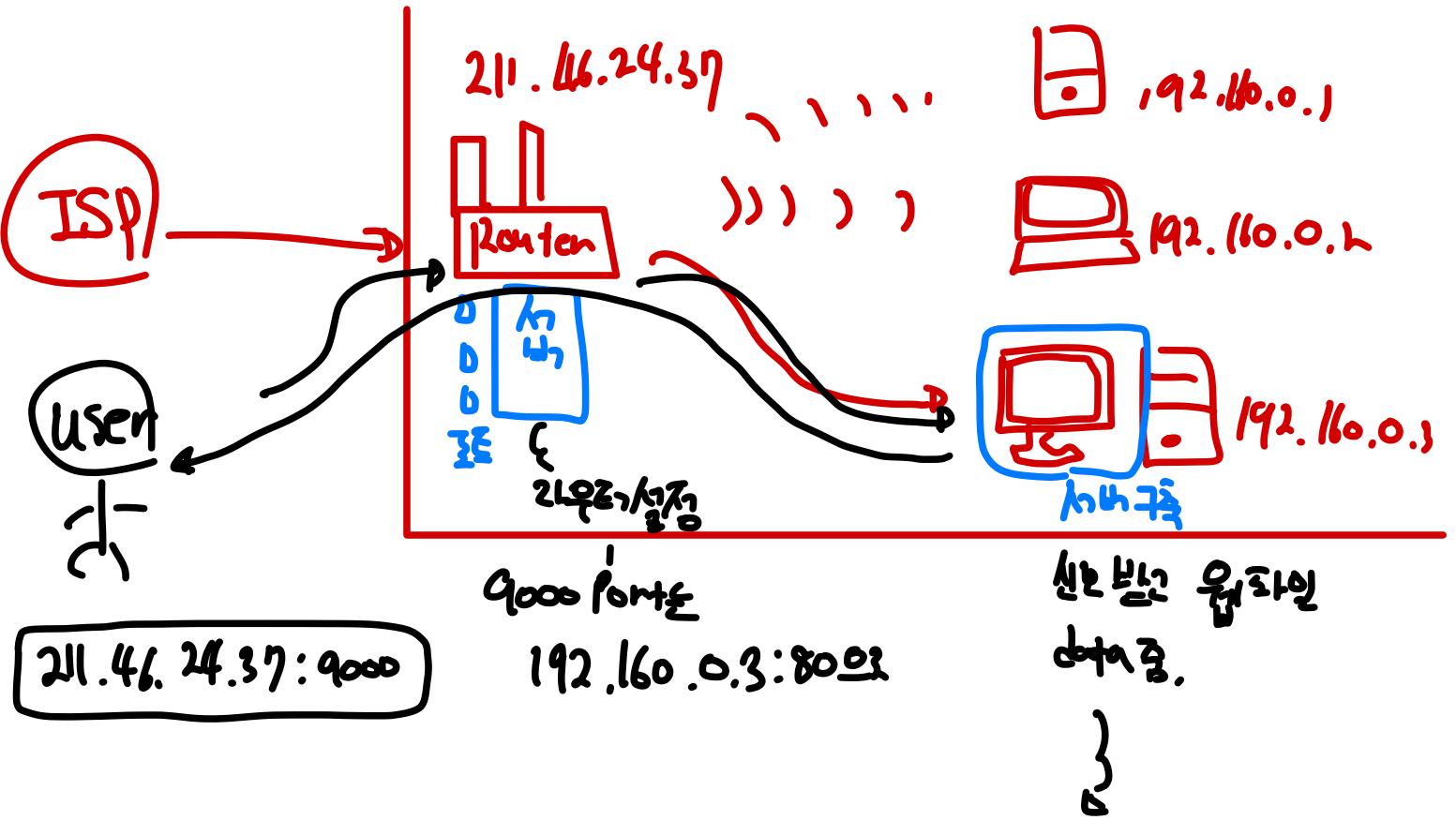
내부 프로그램이 해당 Port에 bind하는 공간

: 추가공간 (상호구별 - Port) - 재미용

NAVER.COM : 80 정식번호 등 . : Web은 80 포트

NAVER.COM : 88 - 80 제외 안정.





Router 가기 위한 경로는?

default Gateway
= 외부로出去 IP 주소

④
인터넷 서버로 쭉 끌어
인터넷 연결하기 위해
파일을 주고 받는 것?
인터넷 - 네트워크

10. 자유SW . 오픈SW



오픈, 풀리 라이선스

Free Software : 비용 X, 자유로운 SW가 많음. (철학) { GNU, GPL }
FSF (Free SW Foundation)

Open Source SW : 소스코드가 Open.

→ PC4로 => 사용 Pool ↑
SW : 과거엔 HW에 종속 → 현재 UNIX : Portability (HW 독관)

· 높은 가치 산업으로 발전.

· 저작권·산권, 라이선스 계약

· 자유SW 핵심 : 자유(free) 용어가 아니라 의의로 통통

영적·물적 GPL 허락 → SW가 빠른 제한
수준

· 오픈소스 등장 ~ Business friendly

소스코드 VS 목적코드

→ 소스코드가 Open되어 Open 소스 SW

○ 소스코드 : 프로그래밍 언어로 작성, 동작원리 쉽게 이해 가능

○ 목적코드 (Object code) 혹은 바이너리 코드 : ex) Reverse engineering

• 컴퓨터 HW 상에서 작업 실행

3

제작자는 개발 X

• 프로그램의 해석, 수정, 편집

• 대형설계 상용 SW는 바이너리화된 제공

자유SW VS 오픈SW

• OS와 오픈SW

- 저작권자가 소스코드를 누구나 특별한 제한 없이 사용, 복제, 배포, 수정 가능 SW (방법론)

• Open Source is a development methodology

- 소스공개 → SW개발을 위한 국제화 (설계, 편집적)



Test

4

장점: [개인]: 빠르고 유연한 개발

오픈소스

→ 첨단 슬라이드

신뢰성, 안전성

↑ 참고

네트워크 지원

AI/ML - OS, 인터넷

• Linux • 아파치 (web서버 50%↑)

:

• BSD

PDF상수

[기업] 로열티 X

언어: Python, Perl, ...

Trend 반영 (초신기술)

Community 활동

개인기반 단속

고객질

투명한 개발 과정

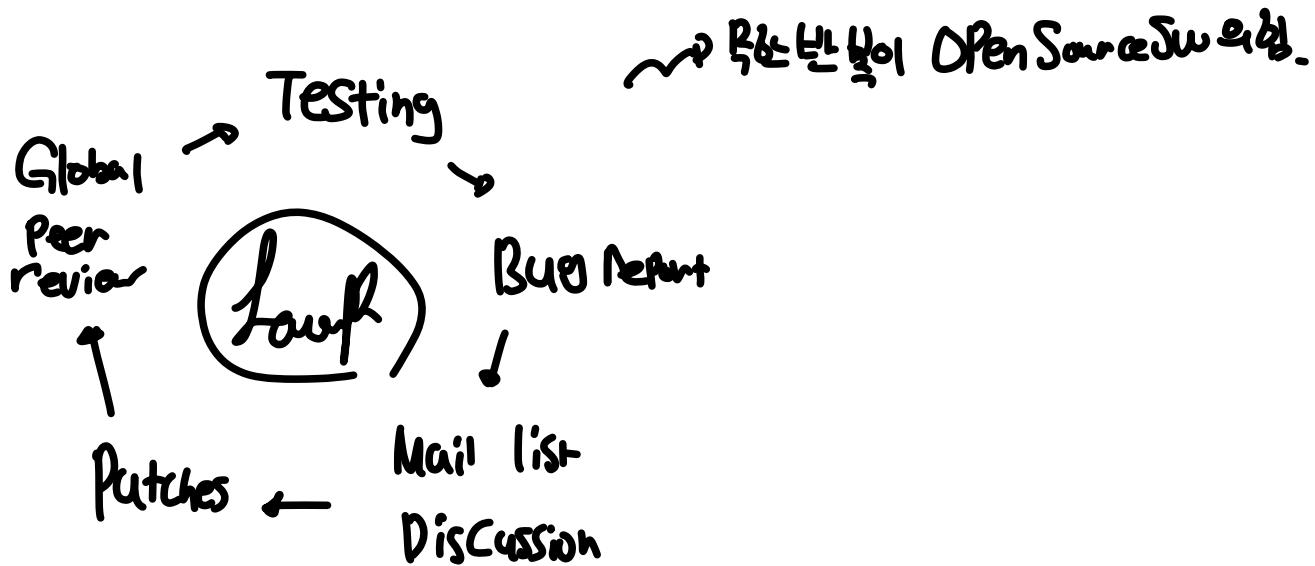
Maintenance 비용 ↓

FSF - FreeSW

이상주의적

SW freedom 중심

• 디자이너가 오픈소스SW를 강화시킬 하는가?



◦ 자유SW 개발 vs 오픈SW 개발



정형화된 개발



自律화된 개발

Top-Down 개발

Bottom-up 개발

Tool 없이는 산만 → VCS (Version Control System)

FSF, GNU Korea: 허브

Open Source Initiative, 공개SW 협회: 네트워크

Freshmeat.net, SourceForge.net: 글로벌

정보사이트.

도메인 등록 PPF가 가능.

II. license : 오픈소스 SW와 관련한 T/F + 추가 궁금?

리눅스는 GPL을ベース로 Open 소스 SW → GPL라이선스 또는 다른 라이선스로 만들면
공개해야 한다.

- SW 저작권 -

- 저작권
- 특허권
- 상표권
- 영업비밀

OSI: 종이 인증: 74개

GPL, LGPL, BSD, 아파치 등이 주로 있음

라이선스 확인 필요

LGPL: 기업 관점

→ declare 저작권 (Copyright 오류)

- 저작권: 프로그래머가 SW 개발시 컴퓨터 프로그램 저작권이 자동 발생
그 권리은 프로그래머 or 그가 속한 회사에 속함.

공개해서 일정기간 보호받고 Open

→ 초기의 특허권을 구속 → 방식(방법)은 대체 허락(상용)

- 특허권: 특허는 특정의 유용하게 하는 방식(Method)이므로 특허권은 방식을
구현하는 SW를 만어, 소스코드가 달라도 해당 특허권과의 명시적인 허락
없이 아예 허락하지 않음.
Not just 기술. 방법임

→ 보관도 명

- 상표권: 상표권을 취득한 SW의 경우 상표를 사용하면서 상표권자의 명시적인 허락
필요

→ Open 안내했다.

- 영업비밀: 공개되지 않은 SW의 경우 영업비밀로 보호 받을 수 있음.

공개된 SW는 이미 다른 부문은 노획으로 볼을 가능성이 있다.

라이선스 : 전반적 허용 (사용허가권)

오픈소스 라이선스 : 사용 방법 및 조건 명시

SW라이선스 - OSS 21이하 모두 가능

- . 일반 상용 SW라이선스: 권리자만 SW의 사용, 복제, 배포, 수정 가능
→ 실행프로그램 형태 (Binary)

- . 오픈소스 SW 라이선스: 사용자의 자유로운 사용, 복제, 배포, 수정 가능

단, 개발자가 정의해 놓은 사용 범위, 조건 엄수

→ 일반적으로 소스코드 + binary (src, bin)

※※※

성직적권 관할 분야 유지: GPL 21이하로 ~ 가능하다

SW의 경우 소스코드와 SW설명, 개발자명, SW버전, 연락처 등 별명을 사용,
등 분야 수정·삭제 안됨.

라이선스 양립성 확인 필요 MPL + GPL
필수: Open, Ex

내용만고 Handbuc PDF 확장 필요

. 광장이나 오픈소스와 Code 공유 가능면 GPL X.

비교

구분	무료이용	배포허용	소스코드취득	소스코드수정	2차저작물 재공개	독점SW와 결합가능
선풍 - GPL	O	O	O	O	O	X
완화 ↓	LGPL	O	O	O	O	O
	MPL	O	O	O	O	O
	BSD License	O	O	O	X	O
	Apache License	O	O	O	X	O
자유 ↴	Freeware	O	O	X	X	X

13. Life Cycle

→ In Unit 1, Object life cycle

◦ SW 개발 생명주기

→ SW engineering : 개발방법론

의미 : SW는 어떻게 개발할 것인가? 이 대안 추산과 같은

순차적 or 병행적 단계 구조

개발모형 or SW공학 모델

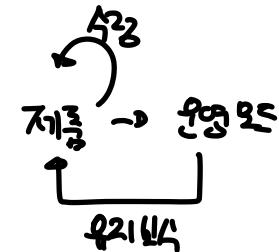
↳ 기획자, 디자이너, 개발자 등 참여자 흥미 존중

* 문서화가 충실히 프로세스 관리를 가능하게 함. (Documentation)

• 모델 예

◦ 주제구식 (Build-Fix Model)

첫 Version



개요 : 요구사항 분석, 설계 단계 없이 일단 개발을 늘어가고 만족도 때까지 반복 시장 수용

크기가 매우 작은 규모의 SW 개발에 훌륭히 일 수 있음. (아날리즈)

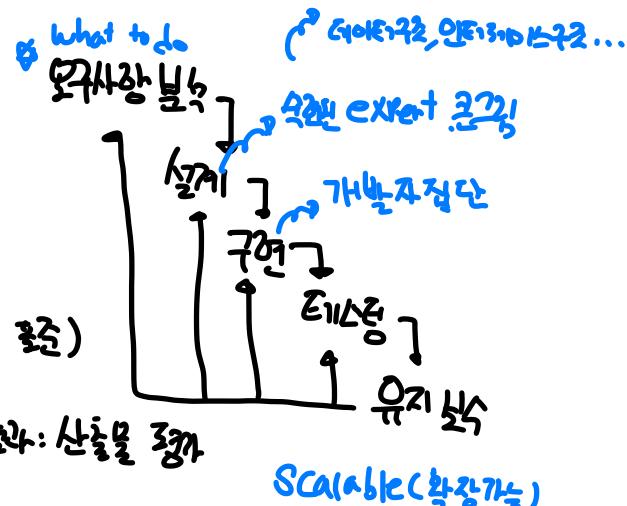
제작 X, 관리자가 전방단계 파악 X, 유지보수 ↓

How to do
여러 알고리즘, 아키텍처,
데이터 구조, 인터페이스 구조...

◦ 폭포식 모델 (Waterfall Model)

순차적으로 SW 개발하는 전통적인 모델 (사실상 업계 표준)

SW 개발 전 과정 나누어 각 단계 진행 → 단계 결과 : 산출물 통합



증강비정형화 + 체계적인 문서화 가능

but 앞단계 완료까지 다음단계 대기, 실제작동 시스템은 개발 후반부에 확인 가능

• 원형 모델

→ Rapid Prototype 후 구체화

원형(Prototype)을 만들어 고객과 사용자가 함께 평가 후 SW 요구사항 정리.

온전한 요구사항 정의 완성.

작동구조: SW 개발 초기에 고속 요구사항 도약 등등

온전히 새로운 Product

시작 → 요구사항 정의

중단 + 인자/기능

원형설계

원형정비

원형개발

고객평가

사용-평가

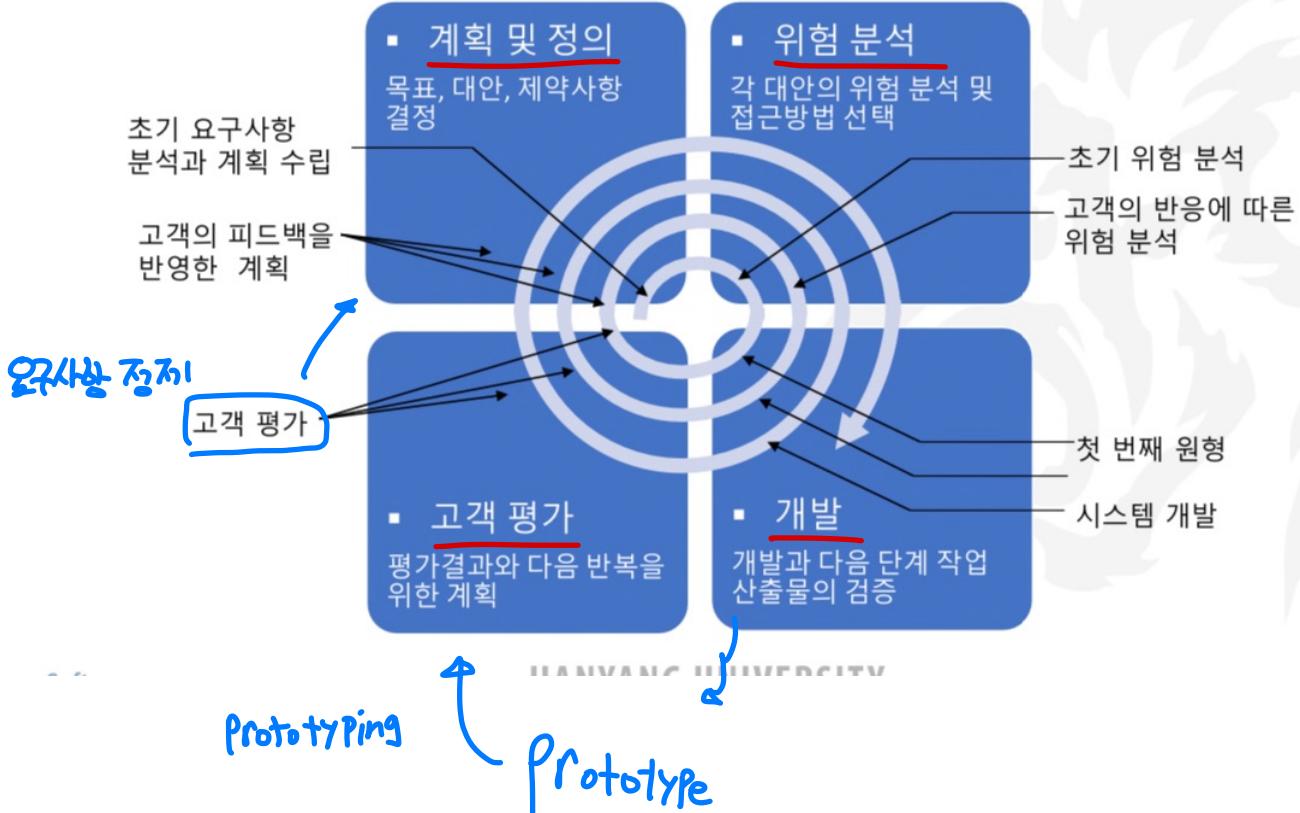
도입, 요구사항 구체화

• 나선형 모델(Spiral Model)

표준 + 원형 + 위험분석 : 점증적 개발모델

{ }
단계별 요구사항
수행 정리

나선형 모델(Spiral Model)



SW개발 생명주기

나선형 모델(Spiral Model)

단계별 활동

- 계획 및 정의 단계
 - 개발자는 고객으로부터 요구사항을 수집
 - 개발자는 시스템의 성능, 기능을 비롯한 시스템의 목표를 규명하고 제약 조건을 파악
 - 목표와 제약 조건에 대한 여러 대안들을 고려하고 평가함으로써 프로젝트 위험의 원인을 규명 가능
- 위험 분석 단계
 - 초기의 요구 사항을 토대로 위험 규명
 - 위험에 대한 평가가 이루어지면 프로젝트를 계속 진행할 것인지 아니면 중단할 것인지를 결정
- 개발 단계
 - 시스템에 대한 생명주기 모델을 선택하거나 원형 또는 최종적인 제품을 만드는 단계
- 고객 평가 단계
 - 구현된 SW(시뮬레이션 모형, 원형 또는 실제 시스템)를 고객이나 사용자가 평가함
 - 고객의 피드백을 얻는데 필요한 작업이 포함
 - 다음 단계에서 고객의 평가를 반영할 수 있는 자료 획득 가능

HANYANG UNIVERSITY

SW개발 생명주기

나선형 모델(Spiral Model)

위험 분석 → 중간에 중단 가능

- 적용 가능한 경우
 - 개발에 따른 위험을 잘 파악하여 대처할 수 있기 때문에
 - ▶ 고비용의 시스템 개발
 - ▶ 시간이 많이 소요되는 큰 시스템 구축 시 유용

장점

- 프로젝트의 모든 단계에서 기술적인 위험을 직접 고려할 수 있어 사전에 위험 감소 가능
- 테스트 비용이나 제품 개발 지연 등의 문제 해결 가능

단점

- 개발자가 정확하지 않은 위험 분석을 했을 경우 심각한 문제 발생 가능

▶ 폭포수, 원형 모델에 비해 상대적으로 복잡하여 프로젝트 관리 자체가 어려울 수 있음

VCS
자원하는 생산성도 되오

-VCS 관련 용어 -

비전번호: 쏘 비전장을 때마다 넣는 고유번호

(7 가지)

분기점(branch): 소스를 중간에 끊고 다른 행위
(수작, 병행 가능)

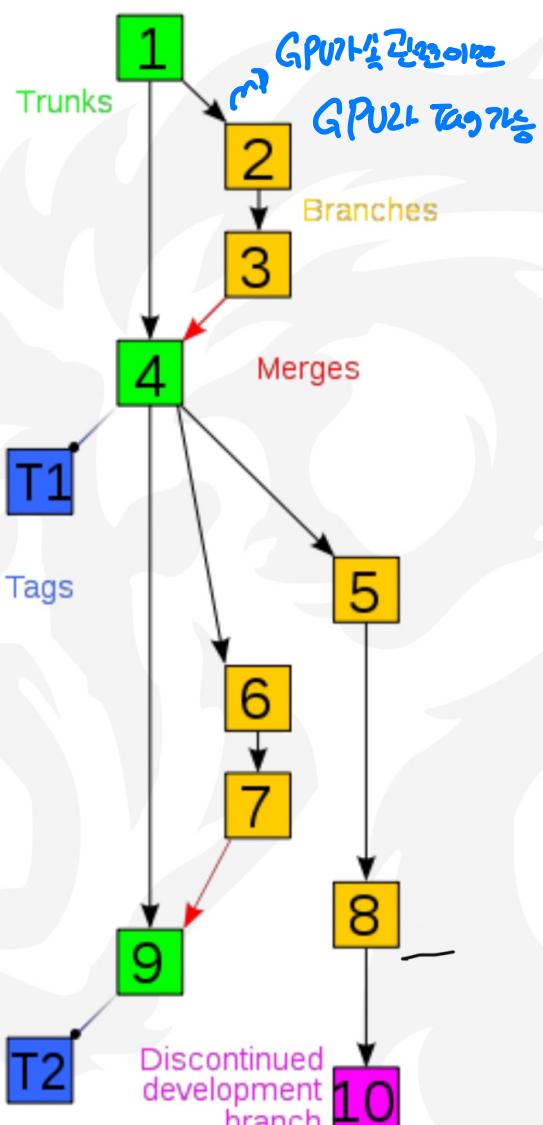
마지(Merge): 두 개의 풀고 싶은 브랜드를 합나다.

방법 아는 학생

본류(trunk): 뼈가 끝나 않은 예를 있는 본류

프로젝트 가장 중심이 되는 디자인

모든 개발 작업은 trunk 브랜치에서 이루어집니다.



리비전(Revision) : 서수정과 내용변경을 위한 과정

특정 시간/보건처의 어떤 한 상태

상대의 핵심은 인적스를 리비전 번호와함

태그(Tag) : 특장한 리비전을 나중에 찾거나

알아보기 위해 불의 텍스트

저장소(Repository)(물리적, 물리적)

코드, 문서, 그림 등을 저장하는 장소

커밋(Commit)

저장소에 변경 사항을 반영하는 행위

커밋 로그(Commit log)

커밋 시, 해당 커밋이 어떤 번들을 했는지 기록

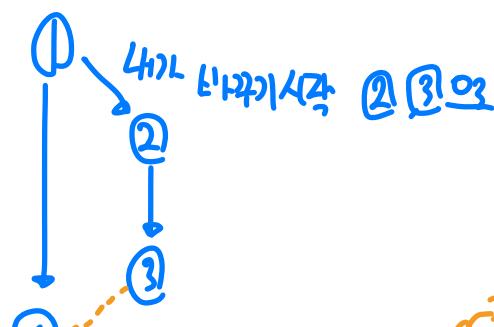
체크 아웃(Check out)

저장소에서 파일 가져오기

체크 인(Check In)

체크 아웃한 파일의 수정이 끝난 경우 저장소에 새로운 버전으로 갱신하는 일

이전에 갱신한지 있을 경우 충돌(Conflict)을 알리면 diff 도구를 이용해 수정하고 커밋하는 과정을 거치게 됨



diff라는 도구로 ④이 합치기 위해 ③을 수정해야 하니 그 책임은 CheckIn하는 ③에게 있다.

파일 가기

Check out

Trunks

2

Branches

Merges

3

T1

Tags

4

Check In

저장소에 다시 갱신
라고 함

실제 반영되는 행위를

Commit이라고 함.

T2

Discontinued
development
branch

5

6

7

8

9

10

VCS

사용 이유: 문의시 놓구 . Roll back

+

누가 무엇을 언제 왜 변경했나?

대규모 수정작업을 안전하게 진행

브랜치 → trunk 보고, 영향최소화 + 새로운 부수 개발 가능

Merge 책임은 Check In하기

트렁크의 history follow 가능

중앙집중식 VCS

|

CVS

분산식 VCS

|

git