Python - HW5

임베디드스쿨1기

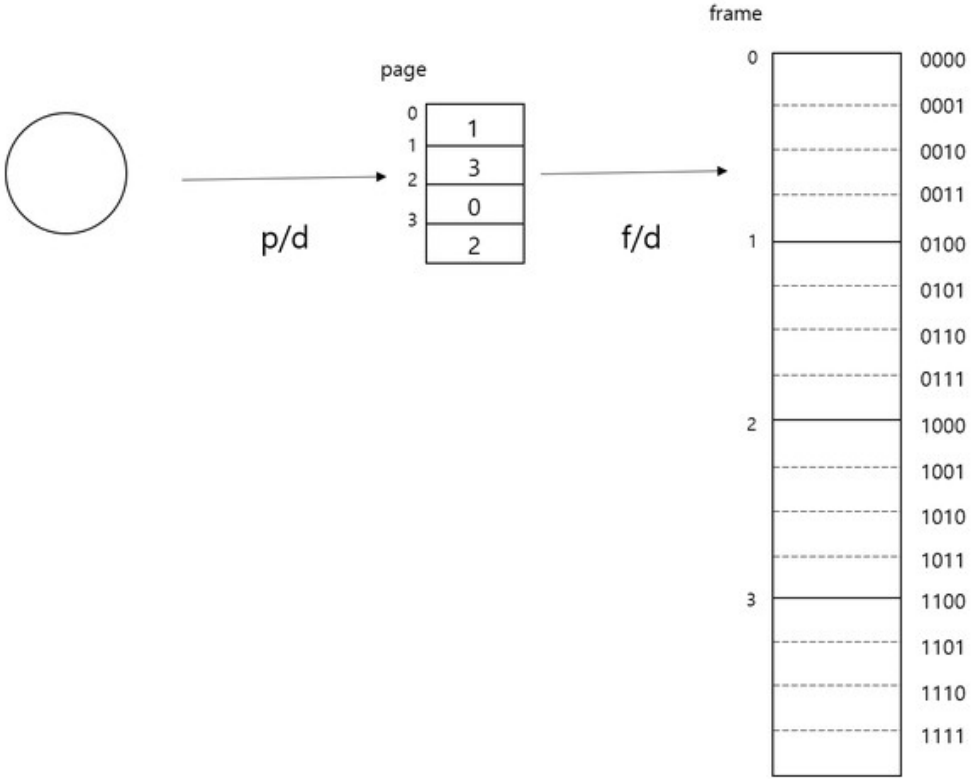
Lv1과정

2020. 08.24

박하늘

1-1. Paging이란?

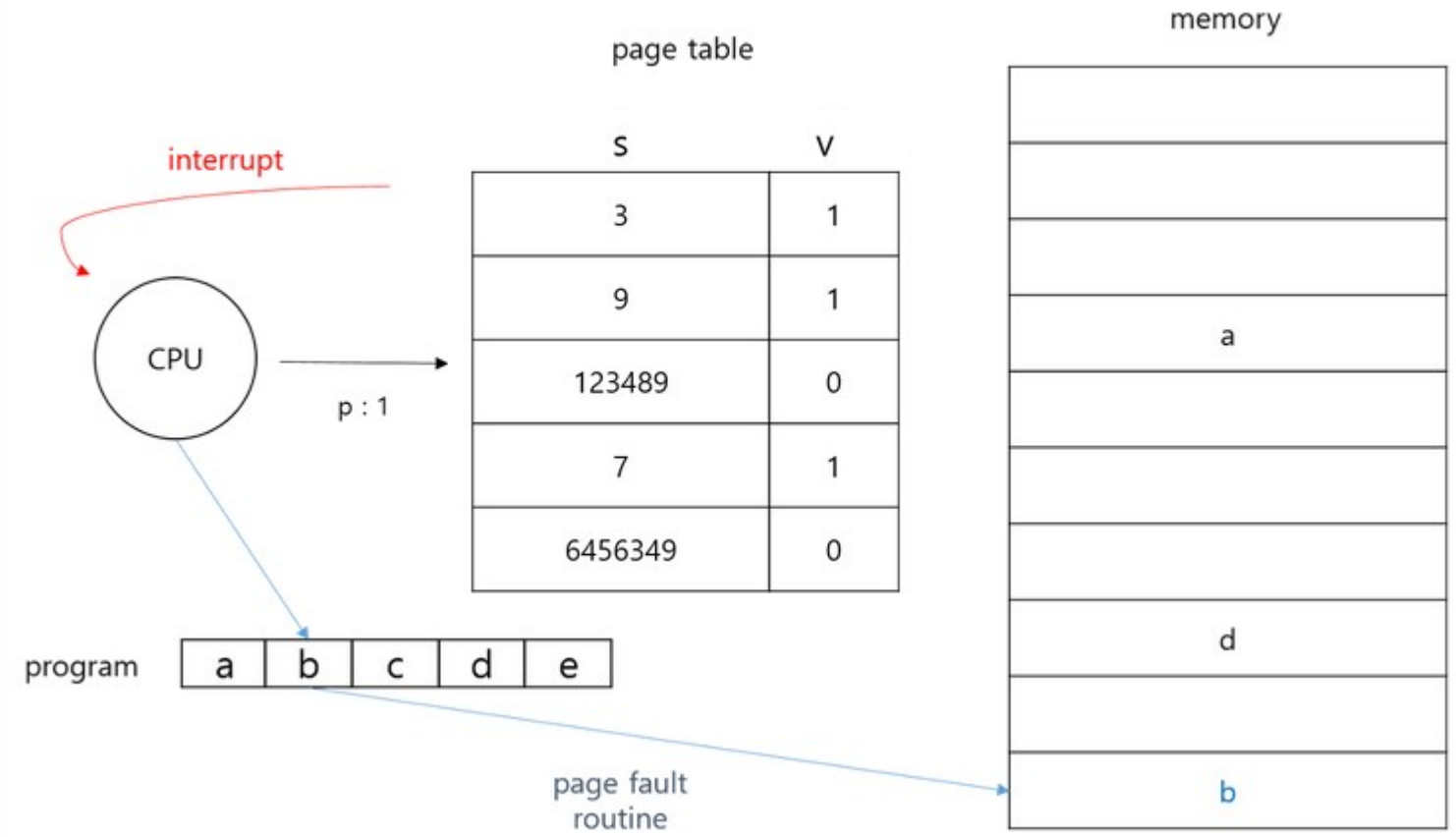
1) MMU에서 Regical address와 physical address의 차이 저장 → 메모리 영역을 일정 크기로 잘게 나눔 → register를 더 두어 그 차이를 저장하여 프로그램 쪼개 적재함 → 외부 단편화 해결.
2) Paging: 프로세스, 논리 주소를 자르는 단위 / Frame: 메모리를 자르는 단위
→ 따라서, 페이지 테이블을 두어 논리 구조가 실제 메모리 안에서는 어느 프레임에 위치하고 있는지 확인가능함



- CPU의 논리 주소는 페이지 테이블을 읽음. 페이지 테이블에서 얻은 프레임 번호 값으로 다시 메모리를 읽음 .
- 페이지 테이블에서 pageNum에 해당하는 데이터를 읽음. 실제 메인 메모리에서 어디에 맵핑할지를 결정하는 프레임 인덱스, frameNum를 얻음.

1-2. Demand Paging이란?

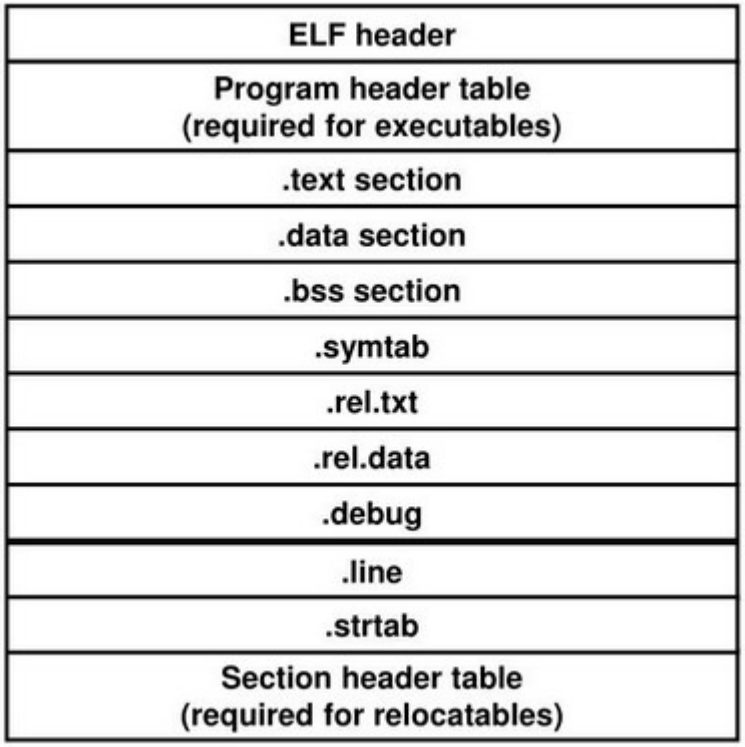
1) 프로세스를 메모리에 적재할 때, 전체 말고 필요한 페이지만 우선 적재. 나머지는 backing stroe에 저장함
2) CPU가 논리 주소를 페이지 테이블에 맵핑했을 때, 기존에 필요했던 페이지에 대한 프레임 인덱스는 페이지 테이블에서 찾음. 등록 되지 않았다 표시된 페이지 프레임 인덱스는 page fault 루틴을 통해 얻음



-테이블 등록된 page: 1
- 등록 안된 page: 0

1-3. Dynamic Linking

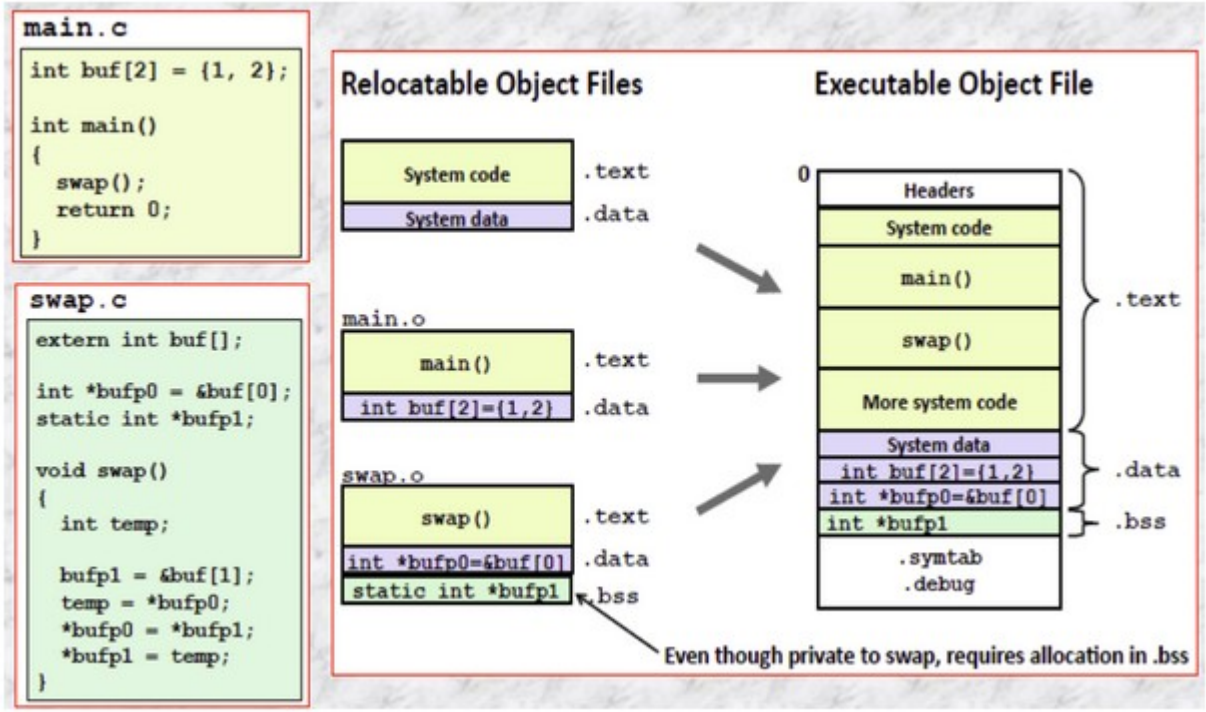
1) 실행파일들의 일반적인 format인 ELF에 대해 알아보자
(ELF: Executable and Linkable Format), 링킹 가능한 파일들: executable, relocatable object file(.o), shared object file(.so)이 모두 따르는 파일 포맷임



- .data 색션에 초기화된 전역 변수 저장됨
- .bss: 초기화되지 않은 전역변수 → 최대한 여야함 , bss는 공간을 차지 하지 않음.
- .rel .text/ .rel .data: relocation에 대한 정보저장(실행파일 만들때 바꿔줘야 하는 포인터들과 인스트럭션의 주소 저장)
- .symtab: symbol table이 저장

- relocation은 컴파일러들이 생성한 relocatable object file(.o 파일)들에 있는 .data와 .text를 빼내 executable file의 .text색션과 .data 색션으로 만드는 것이다.

1-3. Dynamic Linking - 링커 란?



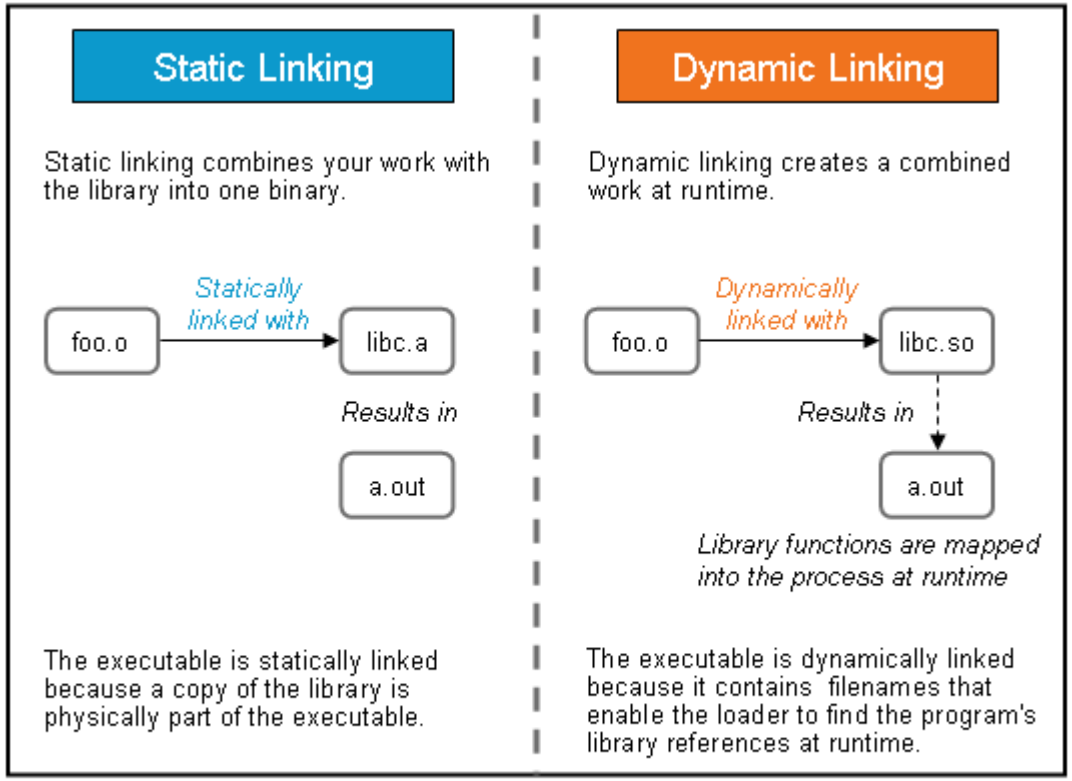
여기서 링킹 없이는 relocatable object files의 .o파일만으로는 실행이 불가능하다. 링커가 수행하는 relocation 때문에 컴파일러는 .o파일을 만들때 어떤 symbol이 어느주소에 있을지 알 수 없다.

결론은 relocation은 컴파일러와 링커의 협력하에 일어난다. 컴파일러는 링커에게 “여기는 링커(너)가 나중에 symbol에 있을 주소를 알아서 적어넣어줘”라는 요청을 보낸다. 링커는 .o파일들을 쪼개서 실행파일을 만든 다음 요청에 따라 적절한 주소를 적어 넣어준다.

1-3. Dynamic Linking – static library 이란 ?

모든 함수들을 하나의 .c로 만들고, .o파일로 만들거나 각각의 함수들을 하나의 소스파일로 만드는 라이브러리 static libraries가 나옴(.lib , a(achive)파일)

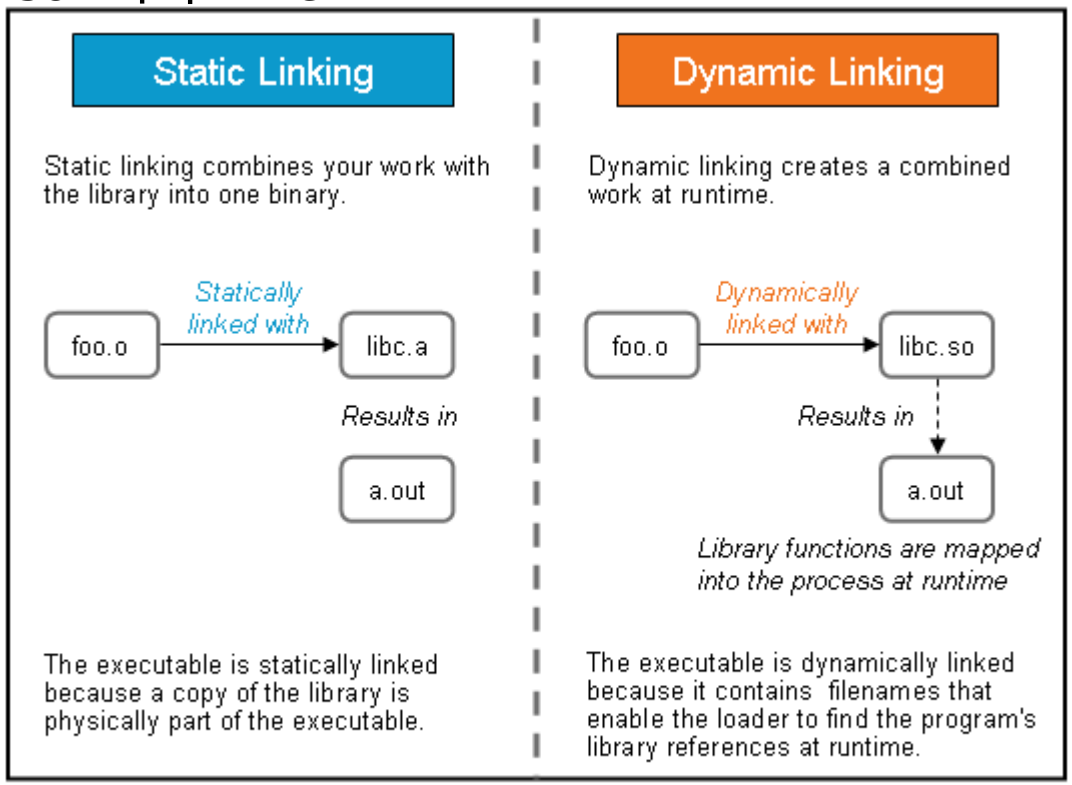
static libraries 단점 발생
- 중복 발생 (printf쓰는 모든 실행파일들 printf.o 파일을 링크해서 가지고 있어야함)
- 현재 실행되는 몇천개의 프로세스가 동시에 실행됨 → 메모리 공간 낭비
- 시스템 라이브러리의 작은 버그 픽스가 일어나면, 모든 프로그램 다시 링크 필요



Dynamic libraries have a "*.so" naming convention and static libraries have an "*.a".

1-4. Dynamic Linking 이란 ?

1)Dynamic libraries 특
- 링킹 동적으로 일어남
(프로그램이 메모리에 올려지는 순간 Load time과 프로그램이 실행중인 순간 run time 동시 사용 가능하다.)
- 실행파일을 만들때 부분적으로 링킹가능하다.
- 프로그램 실행 시에 loader가 실행파일을 메모리에 올린 후, dynamic linker를 호출하고 컨트롤을 넘겨줌
- .so: 리눅스 ver DLL



2-1. 병렬 프로세싱 ILP

1) ILP(instruction Level Paralleliaztion)
- 프로세서가 수행하는 명령어인 Instruction 다수를 동시에 수행하는 병렬
- 명령어 수준 병렬성, 여러 개의 명령어를 병렬로 처리할 수 있는지 정도
- 여러개의 Thread를 동시에 실행해서 여러개의 명령어를 병렬로 처리할 수 있음

2) ILP 프로세서의 연산방법
- 다수의 Instruction을 동시에 처리하는 MISD 혹은 MIMD 수행을 할 수 있다.

〈표 1〉 Flynn's Taxonomy

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

3) 한계
- CPU와 메모리의 상대적인 성능 차이로 인한 Memory wall (메모리 벽)
- 본질적으로 순차적으로 수행 작업이 많아짐
- 깊은 파이프라인으로 실패 비용 발생
- 동시에 사용자 응용프로그램과 운영체제에서 한번에 수행되는 Thread와 Process의 양의 증가하는 추세 → TLP 활용으로 옮겨감

2-2. 병렬 프로세싱 DLP

1) DLP(Data Level Paralleliaztion)

- 여러 데이터를 동시에 처리하는 것.
- 다수의 데이터를 같은 연산의 반복을 통해 처리하므로, 하나의 Instruction으로 동시에 처리함으로써 병렬화

2) DLP 프로세서 연산 방법

다수 데이터의 동시 처리가 가능한 SIMD 혹은 MIMD 수행할 수 있다.

〈표 1〉 Flynn's Taxonomy

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

3) What is Multiprocessor?

메모리를 공유하는 프로세서와 각각 각자의 cache와 memory를 가지고 있는 구조로 나뉜다. 현재 대부분의 컴퓨터는 이 두가지를 적절히 혼합해 사용함.

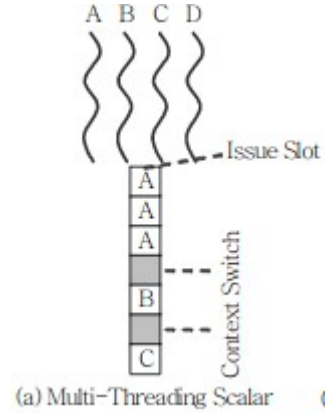
- SISD: 일반적인 uniprocessor와 같은 구조로.한 개 instruction,한 개 data 처리
- SIMD: 한 개 instruction, 여러 개 Data, 동기화를 간소화 함. 고가용성 APP적합
- MIMD: 여러 개 instruction, 여러 개 data처리, message passing machine과 같은 구조, 캐쉬를 보유하지 않기도 함.
- MISD: 여러 개 instruction, 한 개 data, 상당히 비효율적 구조로 인식되어 있음 이론상으론 속도가 매우 빠르다고 하지만 어떤 구조로도 나오지 않음.

2-3. 병렬 프로세싱 TLP

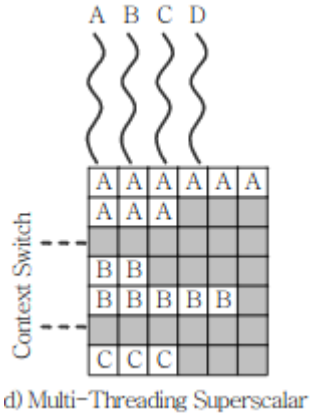
1) TLP(Thread Level Paralleliaztion)
- 프로그램을 수행 흐름의 단위인 Thread로 구성하고, 이들을 동시 수행하여 병렬화
- 스레드 수준 병렬성, 한 번에 몇 개의 스레드를 실행할 수 있는지 정

2) “스레드 수준 병렬 처리”의 강화
- 다수의 스레드를 하드웨어가 실시간으로 관리, 스레드 스위칭시 발생하는 비용 최소화
- 인스트럭션 수행 방법과 스레드 스위칭 방법에 따라 형태가 다양함

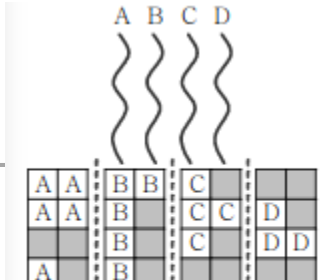
① 하나의 수행 파이프라인을 가짐



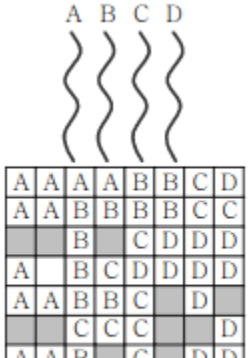
② 다수의 수행 파이프라인을 가짐



③ 수행 파이프라인들이 분리되어, 특정 스레드를 수행하는 형태



④ 다수의 파이프라인이 임의의 스레드 수행

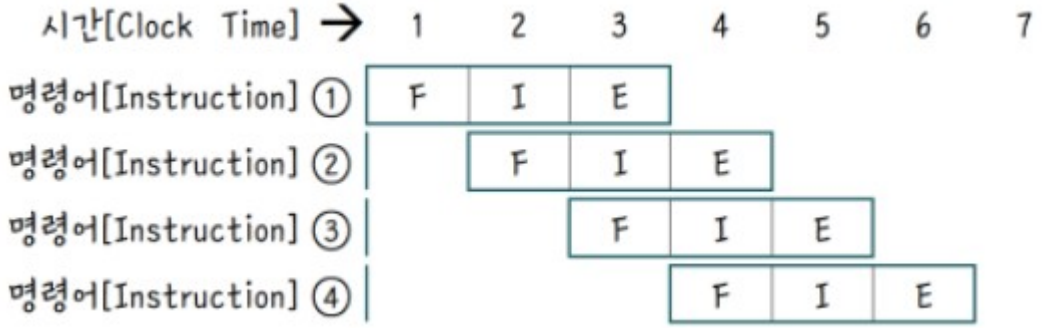


하면 얻는 건 아무것도 없다.

참고. 병렬 컴퓨터 구조의 분류: Flynn 플린 분류

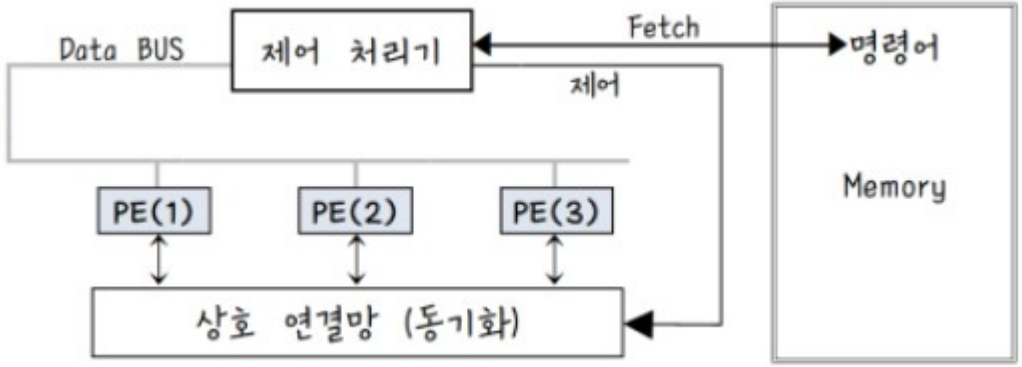
1) SISD

- 하나의 명령어에 하나의 데이터를 처리하는 단일 프로세서 시스템.
- 명령이 하나씩 순서대로 실행됨, 실행 과정은 명령어를 여러 개의 단계로 나누어 파이프라인으로 연결 처리 방식 → 파이프라인 처리기



2) SIMD

- 하나의 명령어에 여러 개의 데이터를 처리하는 배열 처리 형태
- 여러개의 프로세서가 하나의 제어장치에 의해 제어 되는 구조
- 배열처리기



참고. 병렬 컴퓨터 구조의 분류: Flynn 플린 분류

3) MISD
- 하나의 데이터에 대해 여러 개의 명령어를 수행
- 일반 용도의 컴퓨터 구조로는 거의 사용 X
- 백터처리기

4)MIMD
-여러 개의 명령이 여러 개의 데이터를 처리
- 진정한 의미의 병렬 처리 방법
- 다중 처리기 (Multi Processor)

review) 프로그래밍 (파일 읽고 쓰기)

1) Write, Read

- 파일을 열고 난 후에 데이터를 쓰거나 읽어올 때는 read와 write 함수 사용

#include<unistd.h>
ssize_t read(int fd, void *buff, size_t nbytes);
-반환 값: 읽어온 바이트수, 실패 시 -1, 파일의 끝에서 시도하면 0

ssize_t write(int fd, const void *buff, size_t nbytes);
-반환 값: 기록한 바이트 수, 실패 시 -1

read함수를 호출하였을때 반환 값은 읽기 요청한 바이트 수보다 작은 값일 수 있음.(현재 읽어올 데이터 100 바이트뿐인데, 200바이트 읽기 요청하면 100만 읽고 반환)
wirte함수를 호출하면 정확히 요청한 바이트 수 만큼 쓰여지고 반환값도 요청한 바이트 수 야함.