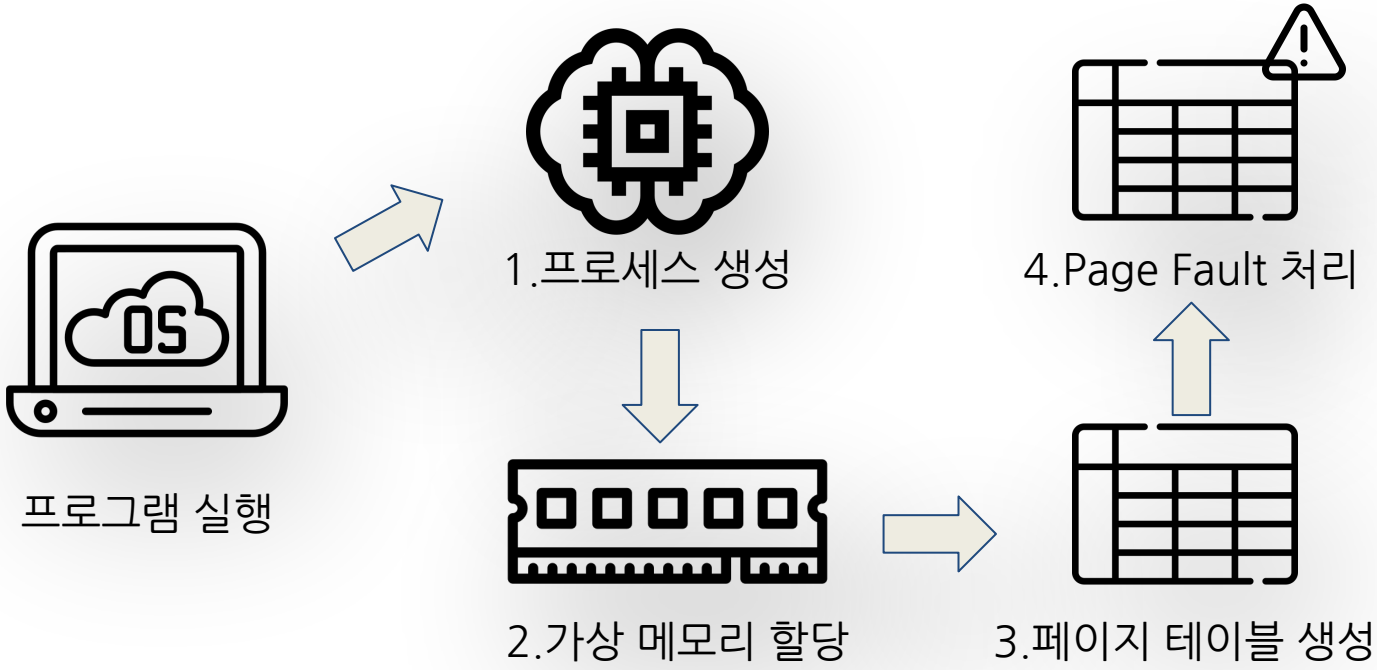
프로세스와 스레드


OS
프로그램 실행
1.프로세스 생성
2.가상 메모리 할당
3.페이지 테이블 생성
4.Page Fault 처리

가상 메모리 주 기억장치 보조 기억장치

프로세스란?

실행중인 프로그램 -> 운영체제에서 실행중인 모든 작업을 나타냄

프로그램 : 실행 가능한 파일이나 코드의 집합
프로세스 : 그 코드가 메모리에서 실행되는 인스턴스

프로세스는 CPU에서 실행 중인 상태와 관련된 모든 정보(메모리, 레지스터, 스케줄링 정보 등)를 포함

프로세스란?

```
park@jaydens-MacBook-Air ~ % ps aux
USER               PID  %CPU %MEM      VSZ    RSS   TT  STAT STARTED      TIME COMMAND
_windowserver      152  44.5  0.7 413053456 113424   ??  Ss   12:43PM 112:29.09 /System/Library/
park              4029  42.1  0.5 412425632  79888   ??  S     3:28PM  57:31.31 /Applications/Go
root               123  24.6  0.1 426956480  15440   ??  Ss   12:43PM   0:21.08 /usr/libexec/ope
park              1025   6.9  1.1 444790000 189472   ??  S    12:43PM  12:10.77 /Applications/Go
root               150   6.9  0.2 426981040  25952   ??  Ss   12:43PM  11:29.57 /usr/libexec/Air
root              1159   5.6  0.2 411246816  26720   ??  Ss   12:43PM   7:45.65 /System/Library/
park               538   5.0  0.6 412526656 107888   ??  S    12:43PM  10:09.99 /Applications/Se
```

프로세스란?

프로세스 ID (PID): 각 프로세스를 식별하는 고유한 숫자

- 운영체제는 PID를 통해 프로세스를 관리하고 추적합니다.

메모리 공간: 프로세스는 실행 중에 필요한 코드, 데이터, 스택 등 메모리 영역을 할당받습니다.

- **코드 영역**: 실행할 프로그램 코드
- **데이터 영역**: 프로그램에서 사용하는 전역 변수, 정적 변수 등
- **힙 영역**: 동적으로 할당되는 메모리
- **스택 영역**: 함수 호출 시 발생하는 지역 변수 및 리턴 주소

프로세스 상태: 프로세스는 여러 가지 상태를 가질 수 있습니다. (예: 실행, 대기, 종료)

스레드란?

프로세스 내에서 실행되는 작은 작업 단위.

하나의 프로세스는 최소 1개에서 여러개의 스레드를 가질 수 있음

스레드와 자원을 공유 -> 같은 가상 메모리 공간을 사용

가상 메모리 주 기억장치 보조 기억장치

스레드란?

Thread ID (TID):

- 각 스레드는 **고유한 스레드 ID (TID)**를 가집니다. 이는 스레드를 구별하는 데 사용됩니다.

2) 스택 (Stack):

- 각 스레드는 자신의 **스택**을 가집니다. 스택은 함수 호출과 지역 변수를 관리하는 메모리 공간으로, 다른 스레드와 **메모리 공간을 공유하지 않습니다**.
- 스택은 함수 호출과 복귀 주소 등을 저장하며, 스레드가 실행되는 동안 동적으로 크기가 변할 수 있습니다.

3) 레지스터 (Registers):

- 스레드는 CPU의 레지스터 상태를 저장합니다. 이는 실행 흐름을 관리하는 데 필수적인 정보를 담고 있습니다.

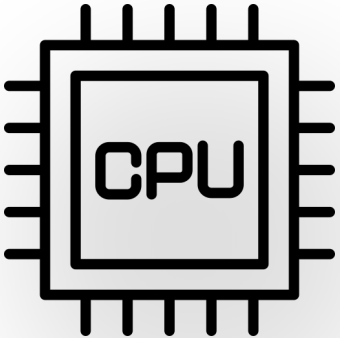
4) 프로그램 카운터 (Program Counter):

- 각 스레드는 자신의 **프로그램 카운터** (PC)를 가지고 있어,자신이 다음에 실행해야 할 명령어의 메모리 주소를 가지고 있습니다. 다른 스레드는 서로 다른 PC 값을 가집니다.

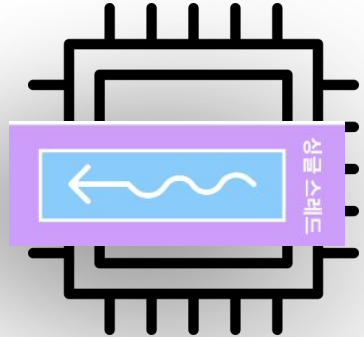
프로세스와 스레드

항목	프로세스	스레드
자원	독립적인 메모리 공간과 자원 할당	프로세스 내에서 자원을 공유
실행 흐름	독립적인 실행 흐름을 가짐	프로세스 내에서 병렬로 실행되는 흐름
상호작용	다른 프로세스와 간섭이 없음	같은 프로세스 내에서만 간섭 가능
생성 비용	상대적으로 높은 자원 소모	낮은 자원 소모 및 빠른 생성 속도
스케줄링	운영 체제에서 독립적으로 스케줄링	프로세스 내에서 스케줄링됨

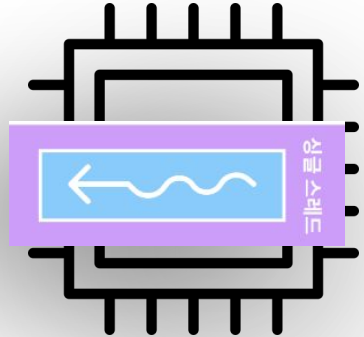
멀티스레싱

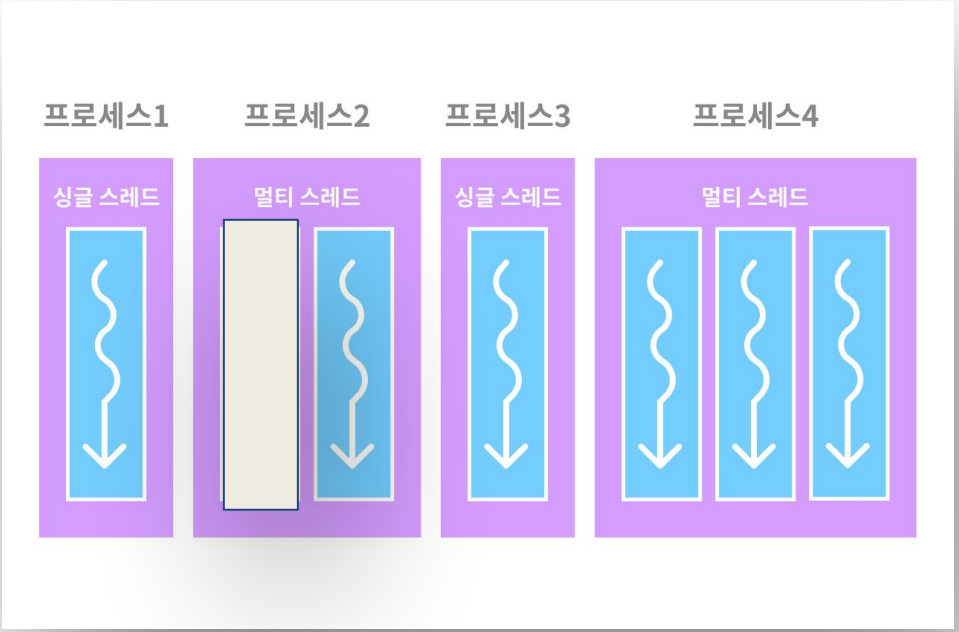


문맥교환 - 시분할

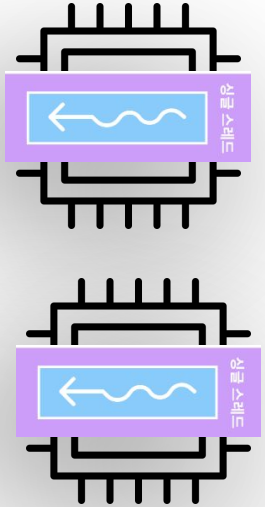


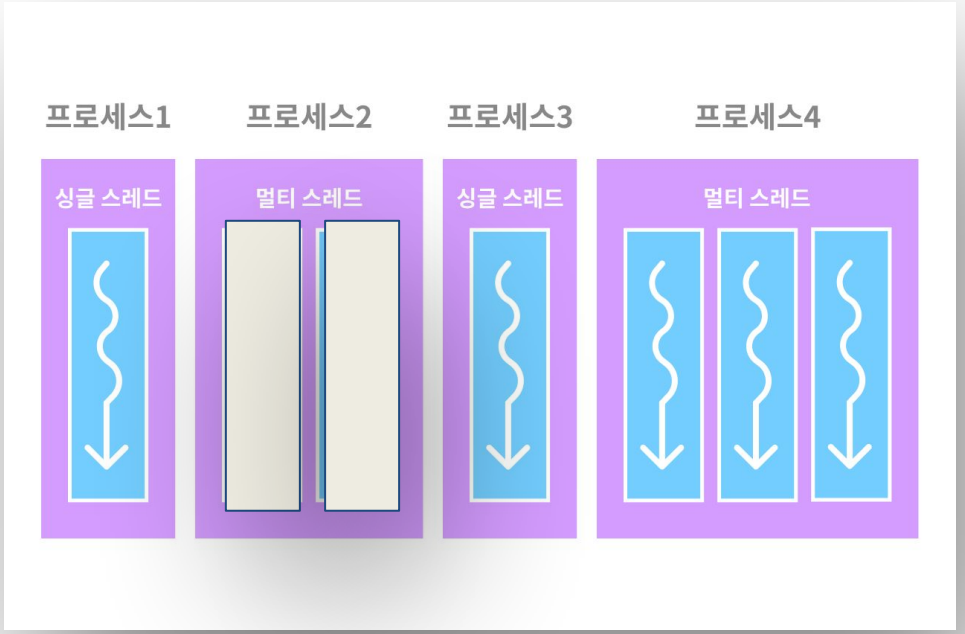
문맥교환



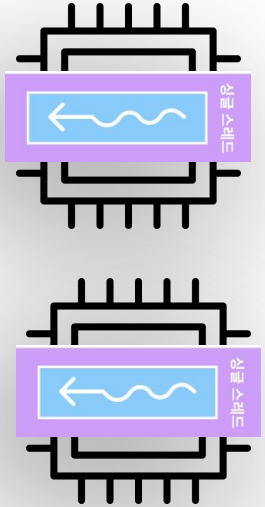


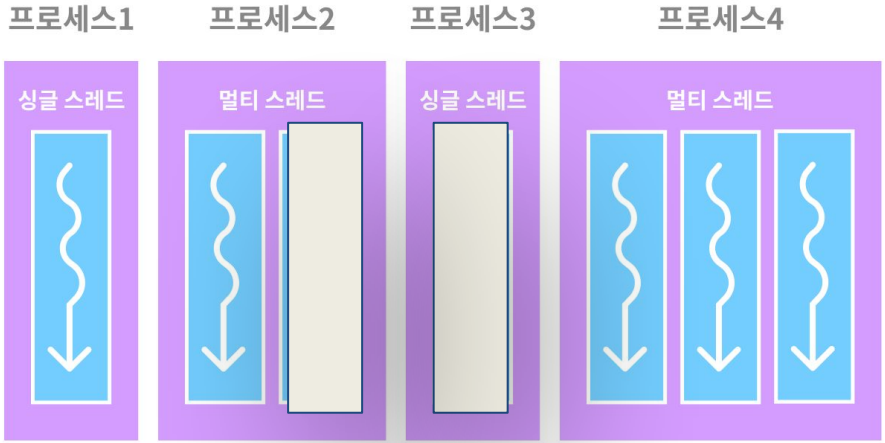
멀티스레싱


싱글 스레드
싱글 스레드

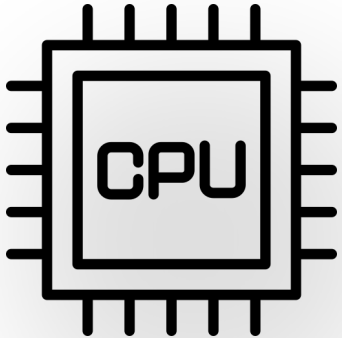

프로세스1
프로세스2
프로세스3
프로세스4
싱글 스레드
멀티 스레드
싱글 스레드
멀티 스레드

멀티프로세싱


싱글 스레드
싱글 스레드


프로세스1
프로세스2
프로세스3
프로세스4
싱글 스레드
멀티 스레드
싱글 스레드
멀티 스레드

프로세스란?



동기화 문제

동기화 문제는 여러 스레드나 프로세스가 공유 자원에 동시에 접근할 때 발생할 수 있는 문제를 의미

여러 스레드가 공유 자원(메모리, 파일, 변수 등)을 동시에 수정하거나 읽으려 할 때, 자원에 대한 접근 순서를 제대로 관리하지 않으면 데이터가 손상되거나 잘못된 결과를 초래

ex) 두개의 쓰레드가 하나의 파일에 접근

멀티쓰레드 상황에서 동기 방식의 접근

프로세스의 동기화

프로세스 간 통신(IPC)은 여러 프로세스가 서로 데이터를 주고받을 수 있도록 해주는 메커니즘입니다. 각 프로세스는 독립적인 메모리 공간을 갖고 있기 때문에, 서로 직접적으로 데이터를 공유할 수 없기 때문에 IPC 메커니즘을 사용.

IPC의 주요 방식:

- 파이프 (Pipe): 한 프로세스의 출력을 다른 프로세스의 입력으로 연결
- 메시지 큐 (Message Queue): 메시지 큐를 사용하여 프로세스 간에 메시지를 전달
- ~~공유 메모리 (Shared Memory): 여러 프로세스가 동일한 메모리 공간을 공유하여 데이터를 주고받음~~
- ~~소켓 (Sockets): 네트워크를 통해 데이터를 주고받는 방법으로, 로컬이나 원격 프로세스 간의 통신이 가능~~

공유 메모리 - 시간 동기화 및 타입 통일이 중요,

스레드간 동기화

뮤텍스와 세마포어

뮤텍스

Mutex (Mutual Exclusion)

- **Mutex**는 상호 배타적 접근을 보장하는 동기화 객체
- 여러 스레드가 동시에 하나의 자원에 접근하지 못하도록 lock과 unlock을 통해 제어
- 하나의 자원에 하나의 쓰레드만 접근 가능

- 사용 예: 은행 계좌에 여러 쓰레드가 접근 하려 할 때, 하나의 쓰레드가 계좌에 접근하여 lock

세마포어

Semaphore

- **Semaphore**는 여러 스레드가 동시에 자원에 접근할 수 있도록 제한하는 동기화 객체
- 일반적으로 **카운팅 세마포어**는 자원에 접근할 수 있는 최대 스레드 수를 지정하고, 이를 초과할 경우 대기하도록 함

사용 예: 한정된 자원(예: DB 커넥션 풀)에 대해 여러 스레드가 동시에 접근할 수 있도록 제한할 때 사용(읽기)

쓰기 작업과 같은 데이터가 변동되는 작업 시에는 뮤텍스와 함께 사용하여 동시성 문제 해결

데드락과 경쟁상태

Dead Lock은 두 개 이상의 프로세스나 스레드가 서로 다른 자원을 점유하고, 각자가 다른 자원을 기다리는 상태. 이로 인해 모든 프로세스나 스레드가 서로 기다리고 있는 상태에 빠지게 되며, 프로그램이 멈춤.

Race Condition은 여러 스레드가 동시에 공유 자원에 접근하고 수정할 때 발생하는 문제. 이로 인해 자원에 대한 올바른 순서가 보장되지 않으면 예상치 못한 결과가 발생 - 뮤텍스와 세마포어로 해결

프로세스와 스레드의 중요성

- **프로세스**는 운영 체제의 주요 단위로서 독립적인 메모리 공간을 할당받아 독립적인 작업을 수행할 수 있어, 안정성과 보안성이 중요한 응용 프로그램에 적합.
- - 웹서버, 데이터 베이스

- **스레드**는 메모리 자원을 공유하여 프로세스의 성능을 더욱 효율적으로 사용할 수 있음. 특히 빠른 응답성과 실시간 처리가 요구되는 애플리케이션에서 중요.
- - 게임 엔진, 파일 다운로드

프로세스란?

```
from threading import
Thread, Lock
import time

# 공유 변수와 락 생성
counter = 0
lock = Lock()

# 작업 함수
def increment():
    global counter
    for _ in
range(100000):
        with lock:  #
락을 사용해 동기화
            counter += 1

if __name__ ==
'__main__':
    # 스레드 생성
    threads =
```

```
from multiprocessing
import Process
import os

# 작업 함수 - 각
프로세스는 독립적인
counter를 증가시킴
def
increment(counter):
    counter += 1
    print(f"프로세스
{os.getpid()}의
counter 값:
{counter}")

if __name__ ==
'__main__':
    # 메인 프로세스에서의
counter 값
    counter = 0
```

프로세스란?