Process Management 1 & 2

상태	완료
생성 일시	@2023년 7월 4일 오후 5:31
수업일	@2023년 7월 4일
유형	정리

프로세스 생성 (부모 프로세스가 자식 프로세스를 만든다.)

- 운영체제가 최초의 프로세스를 생성하면, 이미 존재하는 프로세스가 다른 프로세스를 복제 생성한다. 이때 기존 프로세스를 부모 프로세스라 하고, 새로 생긴 프로세스를 자식 프로세스라고 부른다.
- 프로세스의 트리 (계층 구조)를 형성한다.
- 프로세스는 자원을 필요로 한다(CPU, 메모리 등).
  - 운영체제에게 받는다.
  - 부모와 공유한다(하기도 하고 않하기도 하지만 원칙적으로 공유하지는 않음).
    - 부모 프로세스가 자식 프로세스를 생성하면 별도의 프로세스 이기 때문에 CPU를 얻으려고 경쟁한다.
- 자원의 공유
  - 부모와 자식이 모든 자원을 공유하는 모델
  - 일부를 공유하는 모델
  - 전혀 공유하지 않는 모델(일반적)
- 수행 (Execution) (2가지)
  - 부모와 자식이 공존하며 수행하는 모델 (이때는 부모와 자식이 CPU를 획득하기 위해 경쟁하는 관계가 됨)
  - 자식이 종료(terminate)될 때까지 부모가 기다리는(wait) 모델(Blocked 상태)

즉 부모 프로세스가 자기와 똑같은 자식 프로세스를 하나 만드는 것임 여기서 복제라는 것은 프로세스의 문맥을 모두 복사한다는 것이다. 부모 프로세스의 코,데,힙,스 영역을 그대로 복

사해서 자식 프로세스를 하나만든다 심지어 부모의 PC까지(프로그램이 어디까지 수행 되었는지를 기록하는)

(심화)

자원의 공유 또한 사실 리눅스 같은 OS에서는 코데힙스를 별도로 따로(자식이 카피하기 전에) 만들기 전에 부모의 코데힙스를 그대로 주소를 가리켜 공유하는 구조로 사용하다가 결론적으로는 별도의 프로세스 이기 때문에

각자의 길을 갈때 그제서야 별도의 코,데,힙,스를 카피하여 생성하는 구조이다. 이러한 기법을

copy-on-write(COW기법)라고 한다. : write 가 발생할때 새로운 것을 만들고 그전까지는 그대로 공유하는 것을 말함

주소 공간 (Address space)

- 자식은 부모의 공간을 복사한다. (프로세스 ID를 제외한 운영체제 커널 내의 정보와 주소 공간의 정보)
- 자식은 그 공간에 새로운 프로그램을 올린다.(일단 복제 하고 새로운 프로그램을 덮어씌우는 것이다.)
- 유닉스의 예
  - `fork()` 시스템 콜이 새로운 프로세스를 생성(OS를 통해)한다.
    - 부모를 그대로 복사하고 주소 공간을 할당한다.
  - `fork()` 다음에 이어지는 `exec()` 시스템 콜을 통해 새로운 프로그램으로 주소 공간을 덮어씌운다.
    - 그래서 프로세스 생성은 2단계로 볼 수 있다 무조건 두 함수를 실행하는 것은 아니다 부모 프로세스가 자식을 생성하지 않고 `exec()` 를 통해 자신을 바꿔 버릴 수도 있다.
    - ex 과제 제출 시 이전 ppt를 복사해서 표지 남기고 내용을 바꾸는 것과 유사

프로세스 종료 (Process Termination)

- 프로세스가 마지막 명령을 수행한 후 운영체제에게 이를 알려준다. (`exit()`)
  - 자식이 부모에게 out data를 보낸다.(자식 죽고 부모 죽는 구조이다. 프로세스 세계에서는)

    - 프로세스의 각종 자원이 운영체제에 반납된다.
- 부모 프로세스가 자식의 수행을 (강제)종료한다. (`abort()`)
    - 자식이 할당 자원의 한계치를 넘어선다. (자식이 사치를 많이 부리면 부모가 강제로 죽임)
    - 자식에게 할당된 task가 더 이상 필요하지 않는다. (더 이상 시킬 일이 없을 때 죽임)
    - 부모가 종료(exit)한다. (부모 종료 시 자식의 자식의 자식 즉 스택 구조처럼 생성된 역순으로 차례대로 죽이고 자기가 죽는다.)
        - 운영체제는 부모 프로세스가 종료하는 경우 자식이 더 이상 수행되도록 주지 않는다.
        - 단계적인 종료

`exit()` 는 프로그램이 끝날 때 운영체제에게 자신이 끝났음을 알리는 자발적 종료에 해당하는 시스템 콜이고, `abort()` 는 부모 프로세스가 자식 프로세스의 수행을 중단하는 비자발적 종료에 해당하는 시스템 콜이다.

현실 세계에서는 부모가 자식보다 일찍 죽을 수 있지만 프로세스 측면에서 보았을 때 항상 원칙이 있는데 자식프로세스가 먼저 죽어야 한다. 그다음 부모가 뒤처리하는 구조이다.

각종 시스템 콜

- 프로세스와 관련한 시스템 콜 4가지
    - ***`fork()`***, ***`exec()`***, ***`wait()`***, `exit()`

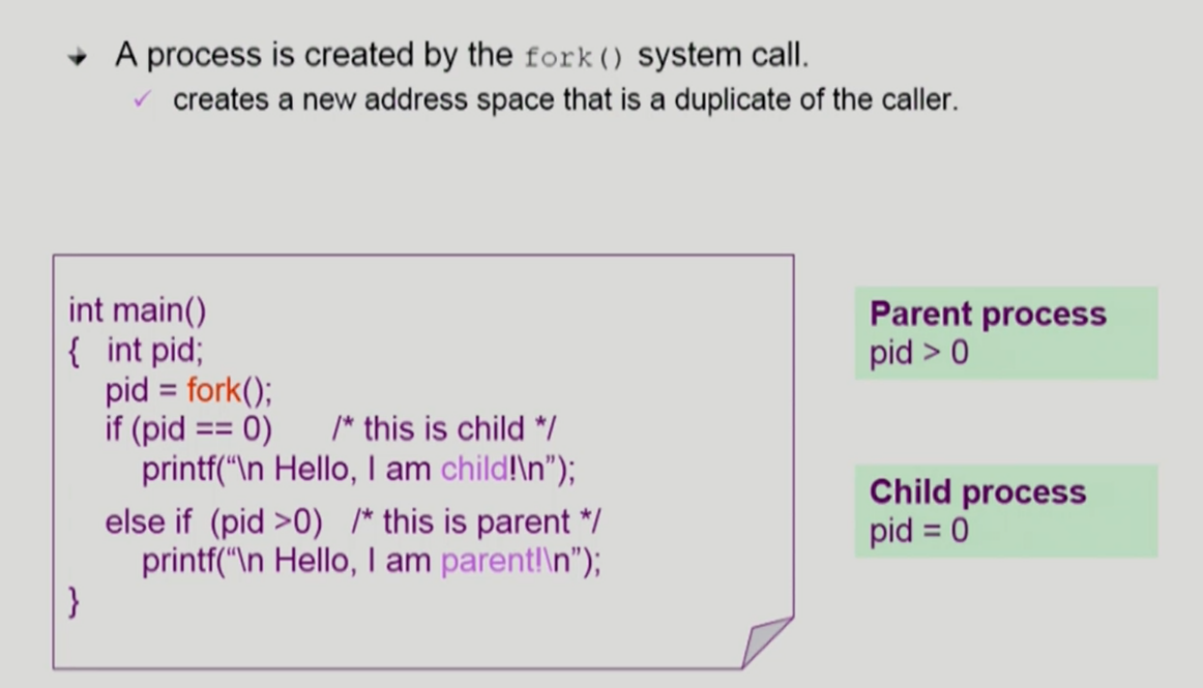
***`fork()`* 시스템 콜**

개념

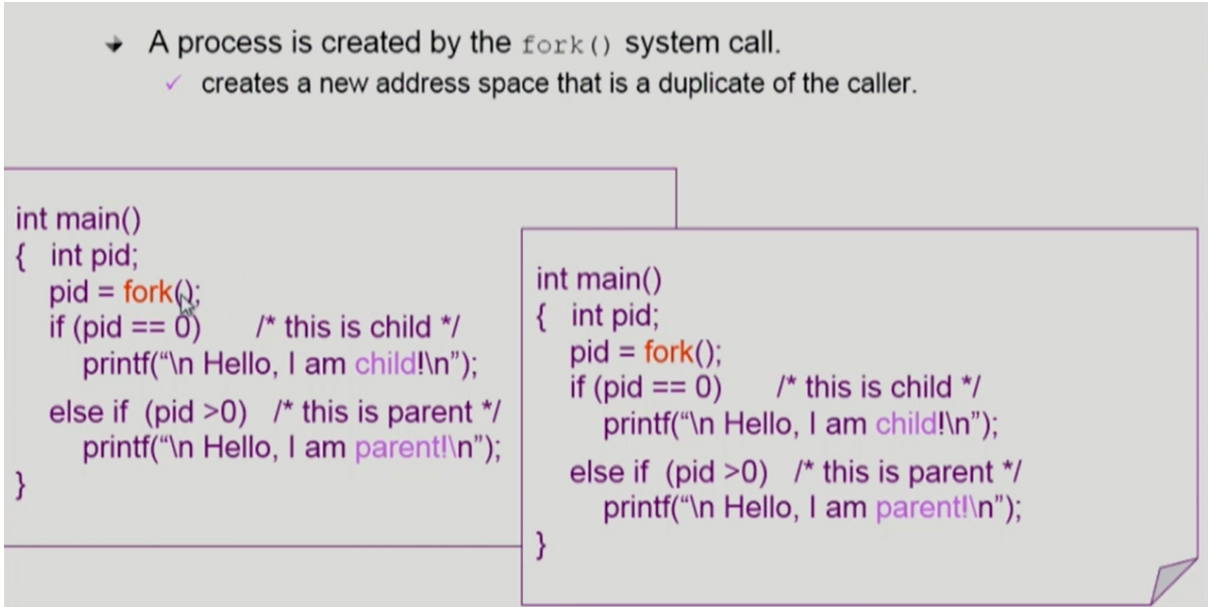
- 운영체제는 자식 프로세스의 생성을 위해 `fork()` 시스템 콜을 제공한다.
- 프로세스가 해당 시스템 콜을 호출하면 CPU의 제어권이 커널로 넘어가고, 커널은 `fork()` 를 호출한 프로세스를 복제하여 자식 프로세스를 생성한다.

- `fork()` 를 수행하면 부모 프로세스의 주소 공간을 비롯해 프로그램 카운터 등 레지스터 상태, PCB 및 커널 스택 등 모든 문맥을 그대로 복제해 자식 프로세스의 문맥을 형성한다.
- 따라서 자식 프로세스는 부모 프로세스의 처음부터 수행하지 않고, 부모 프로세스가 현재 수행한 시점부터 수행하게 된다.
- 다만 자식 프로세스와 부모 프로세스의 식별자는 다르다. (운영체제가 프로세스를 관리해야 하기 때문이다.)

소스 코드



- 부모 프로세스가 메인 함수의 첫 번째 줄부터 한 줄씩 코드를 수행하다가 `fork()` 라인에 이르면 자신과 똑같은 프로세스를 하나 생성한다.



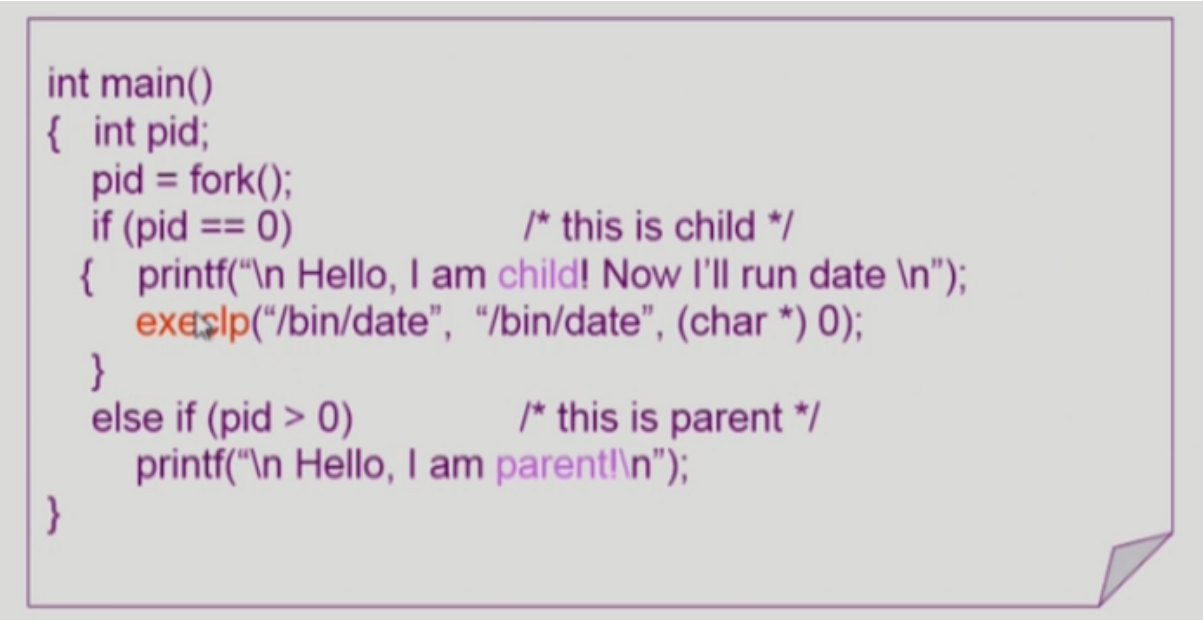
- 이때 `fork()` 라인까지 수행했다는 기억조차도 똑같은 자식 프로세스가 생성된다.
  - 문맥을 복사해서 생성하니까 즉 PC : 어디까지 수행했었는지도 복사가 되는 것이다.
- 그래서 복제된 프로세스는 자기가 복제본이 아니라 원본이며, 자기를 복제해서 다른 복제본을 생성했다는 기억을 갖게 된다. (심지어 이전에 코드들도 실제로 실행한 적이 없지만 수행했었지.. 라는 기억이 있다.)
- 다만 이 자식 프로세스가 복제된 프로세스라는 사실을 알 수 있는 단서가 있는데, `fork()` 의결과 값으로 0을 반환한다. (진짜 부모 프로세스는 0보다 큰 값을 호출한다.)

`exec()` 시스템 콜

개념

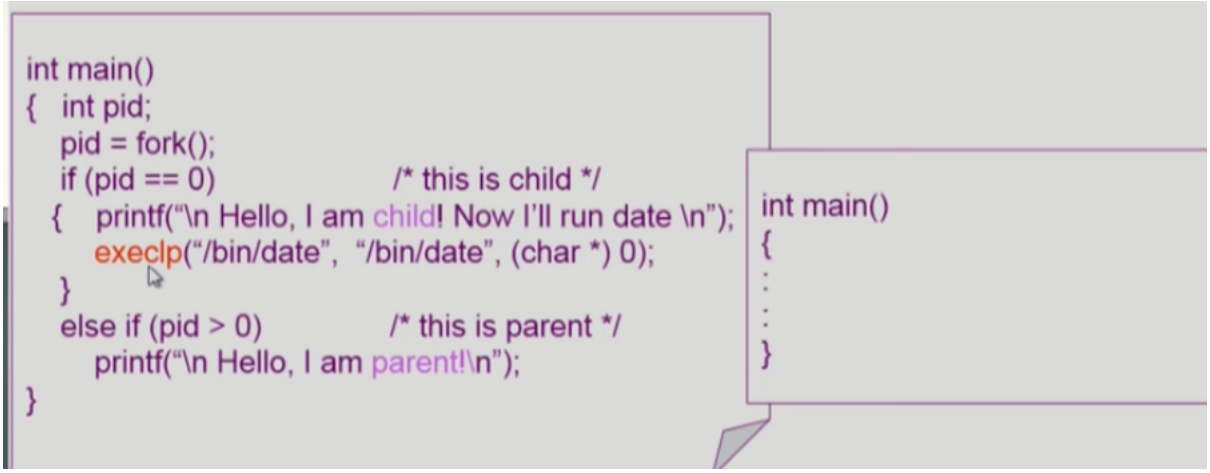
- `fork()` 시스템 콜만으로는 같은 코드에 대해 조건을 분기하는 정도로 밖에 사용할 수 없다.(부모면 A 자식이면 B)
- 자식 프로세스에게 부모 프로세스와는 독자적인 프로그램을 수행할 수 있는 메커니즘이 바로 `exec()` 시스템 콜이다.
- 이 시스템 콜은 프로세스의 주소 공간에 새로운 프로그램을 덮어 씌운 후, 새로운 프로그램의 첫 부분부터 다시 실행하도록 한다.
- 따라서 새로운 프로그램을 수행하기 위해서는 `fork()` 시스템 콜로 복제 프로세스를 생성한 뒤, `exec()` 시스템 콜로 해당 프로세스의 주소 공간을 새롭게 수행하려는 프로세스의 주소 공간으로 덮어 씌우면 된다.

소스 코드



```
int main()
{   int pid;
    pid = fork();
    if (pid == 0)                    /* this is child */
    {    printf("\n Hello, I am child! Now I'll run date \n");
         execlp("/bin/date",  "/bin/date", (char *) 0);
    }
    else if (pid > 0)                /* this is parent */
         printf("\n Hello, I am parent!\n");
}
```

- `execlp()` 라인에 이르면 새로운 프로그램으로 덮어 씌운다.
  - 그러면 지정해준 경로의 새로운 프로그램의 main부터 실행 하는 것(순수한 기억을 가지고)이다.



```
int main()
{   int pid;
    pid = fork();
    if (pid == 0)                    /* this is child */
    {    printf("\n Hello, I am child! Now I'll run date \n");
         execlp("/bin/date",  "/bin/date", (char *) 0);
    }
    else if (pid > 0)                /* this is parent */
         printf("\n Hello, I am parent!\n");
}
```

```
int main()
{
:
:
}
```

나는 자식이야 라는 문자열을 출력하고 `execlp()` 을 통해 우측에 있는 새로운 프로그램을 처음부터 다시 실행 하는 구조이고 `execlp()`을 통해 넘어간 순간 다시 돌아올 수 없다.

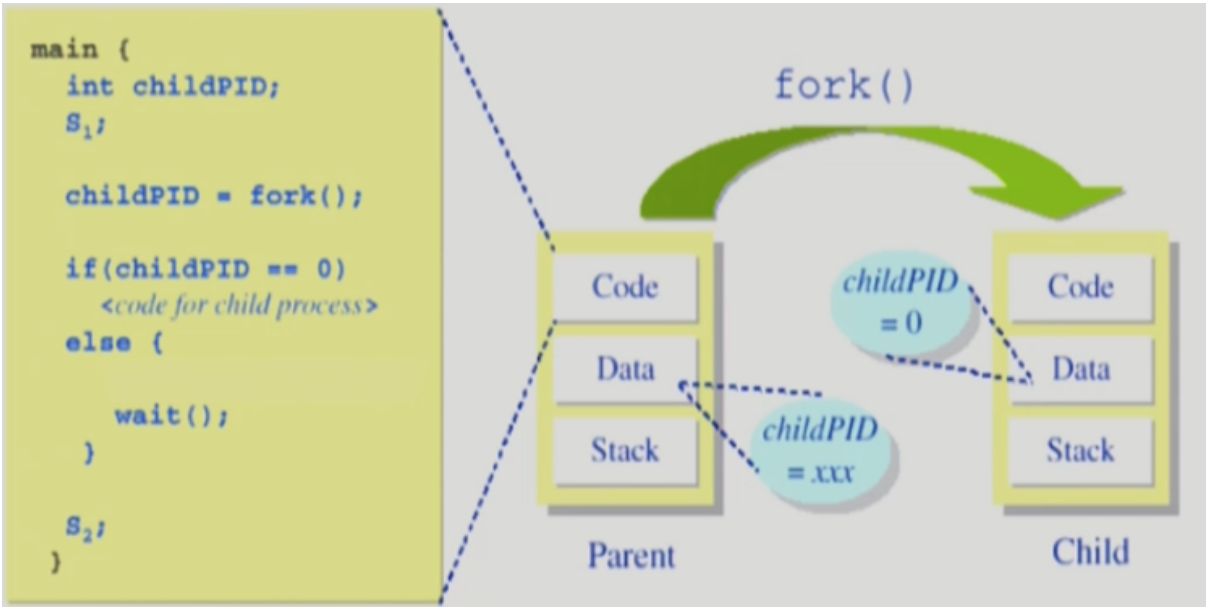
추가로 자식을 생성하지 않고 그냥 사용할 수 있는데 이럴 경우 아래

나는 부모야라는 문자열은 출력될 일이 없다.

```
int main()
    printf("\n Hello, I am child! Now I'll run date \n");
    execlp("/bin/date",  "/bin/date", (char *) 0);
    printf("\n Hello, I am parent!\n");
}
```

`wait()` 시스템 콜

- 자식 프로세스가 종료되기를 기다리며 부모 프로세스가 봉쇄 상태에 머무르도록 할 때 사용한다.
- 부모 프로세스가 `fork()` 후에 `wait()` 을 호출하면 자식 프로세스가 종료될 때까지 부모 프로세스를 봉쇄 상태(block 상태)에 머무르게 하고, 자식 프로세스가 종료되면 부모 프로세스를 준비 상태(ready 상태)로 변경한다.

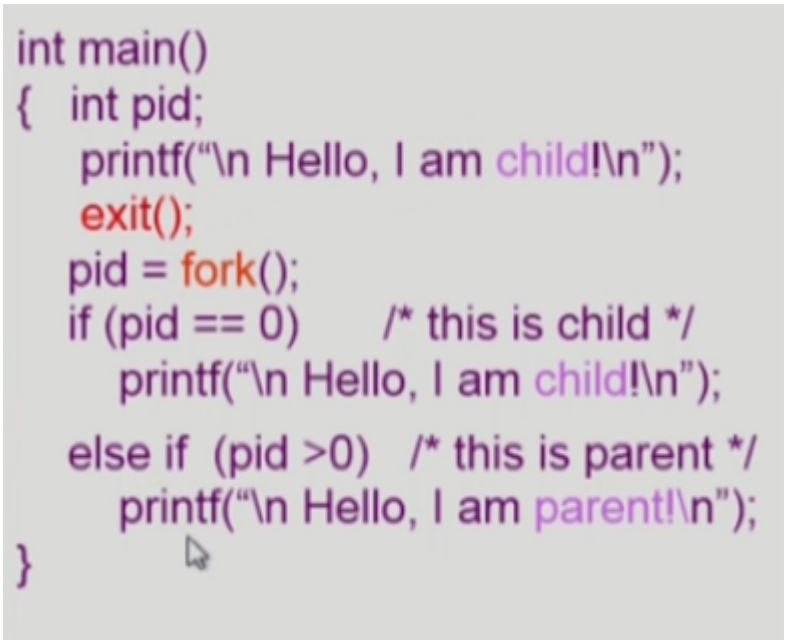


fork 이후 자식이라면 자식 코드를 실행하면 되고

내가 부모 프로세스일 때는 wait()를 통하여 sleep 즉 잠시 block 상태에서 대기 한다 자식 프로세스가 종료될 될 때까지 위에서 정리했던 것처럼 프로세스 생성 수행 방법의 하나임

exit() 시스템 콜

- 프로세스를 자발적으로 종료할 때 사용한다.
  - 죽은 다음 부모 프로세스에게 알린다.
- 마지막 statement 수행 후 `exit()` 시스템 콜을 수행한다.
- 프로그램에 명시적으로 적어주지 않아도 main 함수가 리턴되는 위치에 컴파일러가 넣어준다.



해당 코드에서는 처음 문자열만 출력하고 바로 종료 되버리는 것이다.

`abort()` 시스템 콜

- 프로세스를 비자발적으로 종료할 때 사용한다.
- 부모 프로세스가 자식 프로세스를 강제로 종료한다.
  - 자식 프로세스가 한계를 넘어서는 자원 요청
  - 자식에게 할당된 task가 더 이상 필요하지 않음
- 키보드로 kill, break 등을 입력한 경우
- 부모가 종료하는 경우
  - 부모 프로세스가 종료하기 전에 자식들이 먼저 종료됨.
    - 계층적으로 죽어서 부모가 마지막으로 죽음

프로세스 간의 협력

독립적 프로세스 (Independent process)

- 프로세스는 각자의 주소 공간을 가지고 수행되므로 원칙적으로 하나의 프로세스는 다른 프로세스의 수행에 영향을 미치지 못한다.

협력 프로세스 (Cooperating process)

- 경우에 따라서 프로세스 협력 메커니즘을 통해 하나의 프로세스가 다른 프로세스의 수행에 영향을 미칠 수 있다.

프로세스 간 협력 메커니즘 (IPC: Interprocess Communication)

- IPC는 프로세스들 간의 통신과 동기화를 이루기 위한 메커니즘을 의미한다.
- 메시지 전달 방식 (Message Passing : 커널을 통해 메시지 전달)과 공유 메모리 방식 (shared memory : 일부 주소 공간을 공유)이 있다.

메시지 전달 방식 (Message Passing) (2가지)

- 프로세스 간에 공유 데이터를 일체 사용하지 않고 메시지를 주고 받으면서 통신하는 방식이다.
- 메시지 통신을 하는 시스템은 커널에 의해 send와 receive 연산을 제공받는다.

아래 처럼 두가지로 나뉘긴 하지만 두가지 다 메세지를 전달 하기위해 커널을 통해 전달 한다.

1. **직접 통신 (Direct Communication)**

- 통신하려는 프로세스의 이름을 명시적으로 표시한다.
- 각 쌍의 프로세스에게는 오직 하나의 링크만 존재한다.

간접 통신 (Indirect Communication)

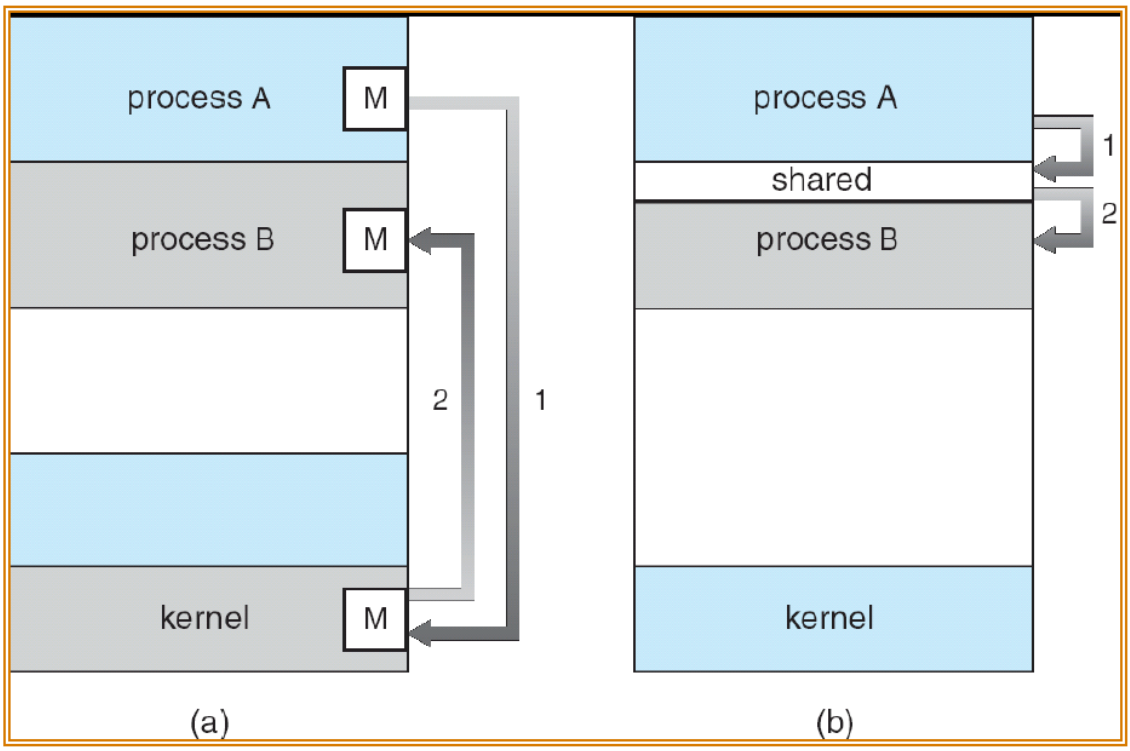
- 메시지를 메일 박스 또는 포트로부터 전달받는다. (누가 꺼내볼지는 명시하지 않음)
- 각 메일 박스에는 고유의 ID가 있으며 메일 박스를 공유(하나가 아닌 여러 개가 전달받을 수 있는 것이다.)하는 프로세스만 통신할 수 있다.
- 하나의 링크가 여러 프로세스에 할당될 수 있다.

둘 다 아래 그림(왼쪽)처럼 커널이 먼저 받고 커널에 의해 전달받는다.

공유 메모리 방식 (Shared Memory)

원칙적으로 프로세스들은 독자적인 주소공간을 가지고 있어서 코,데,힙,스 영역을 각자 가지고 있고 자기 주소 공간만 볼 수 있는데 ***Shared Memory는 일부 주소공간을 공유하는 방법이다.***

아래 그림(오른쪽)처럼 직접적으로 공유 할 수 있는 것이 아니라 커널에 ***Shared Memory*** 를 쓴다는 시스템 콜을 해서 매핑을 해놓고 그때부터 사용자 프로세스끼리 해당 영역을 공유하는 것이다.



Message Passing Shared Memory

✓ Observe: in a distributed system, message-passing is the only possible communication model.

- 공유 메모리 방식은 프로세스들이 주소 공간의 일부를 공유한다.
- 서로의 데이터에 일관성 문제가 유발될 수 있다.
- 프로세스 간 동기화 문제를 스스로 책임져야 한다.

---

thread: thread(여러 개여도)는 사실상 하나의 프로세스이므로 프로세스 간 협력으로 보기는 어렵지만 동일한 process를 구성하는 thread들 간에는 주소공간을 공유하므로 협력이 가능하다.