4. Process Management

프로세스 생성

- 부모 프로세스가 자식 프로세스를 생성
  - 자식을 복제해서 생성 (프로세스의 문맥을 복사)
  - 트리 형태로 형성
- 프로세스가 실행이 되려면 자원이 필요 (메모리 , CPU)
  - 운영체제로부터 받음
  - 자원을 공유하는 경우도 있고 아닌 경우도 있음. (원칙은 아님)
    - 보통 자원을 놓고 경쟁 (별개의 프로세스)
    - 리눅스. 주소 공간 공유하면서 프로그램 카운터만 카피
      - Copy-On-Write(COW)
        - write가 발생 했을 때 카피 (원래 있던 내용이 바뀜)
- 수행
  - 부모, 자식이 공존하며 수행되는 모델
  - 자식이 종료 할 때까지 부모가 기다리는 모델
- 주소 공간
  - 자식 프로세스가 부모의 주소 공간을 복사 (binary and OS data)
    - 자식은 그 공간에 새로운 프로그램을 올림
  - 유닉스
    - fork() -> 새로운 프로세스 생성(복사 + 주소공간 할당)을 요청하는 시스템 콜
    - exec() 시스템 콜을 통해 새로운 프로세스를 메모리에 올림
    - exit : 프로세스가 마지막 명령 수행한 후 운영체제에 이를 알림
      - 자식이 부모에게 output data 보내게 됨. (wait)
      - 프로세스의 각종 자원들이 운영체제에게 반납됨.
    - abort : 강제 종료. 부모가 자식의 수행 종료 시킴
      - 자식이 할당 자원 한계치를 넘어섬
      - 자식이 더 이상 필요하지 않음
      - 부모 프로세스가 종료

프로세스와 관련된 시스템콜

fork()

```
int(main)

{ int pid;

pid = fork();

if (pid == 0 ) /* this is child */
```

```
printf("\nHello,, I am child!\n*);

else if (pid > 0) /* this is parent */

printf("\n Hello, I am Parent!\n");

}
```

- 부모는 fork 이후 아무렇지 않게 아래를 실행
- 새로운 프로세스는 main 함수부터 실행하는 게 아님
  - fork 아래 부터 실행 (pid까지 실행된 것도 복사됨)
- 운영 체제는 복제 이후 자식과 부모 구분
  - 부모 프로세스는 결과값으로 양수를 받음
    - else if 부분 실행
  - 자식 프로세스는 0을 받음
    - if 부분 실행

exec()

- 어떤 프로그램을 완전 새로운 프로세스로 만드는 역할

```
int(main)

{ int pid;

pid = fork();

if (pid == 0 ) /* this is child */

{
printf("\nHello,, I am child!\n*);
execlp("/bin/date", "/bin/date", (char*)0; /* date라는 새로운 프로그램 */
}

else if (pid > 0) /* this is parent */

printf("\n Hello, I am Parent!\n");

}
```

- exec() 시스템 콜을 만나면 지금까지 기억 잃고 완전 새로운 시스템으로 태어남
- exec() fork 시에만 하는 것 아님.
  - ```
    int(main)

    { int pid;

    pid = fork();

    if (pid == 0 ) /* this is child */

    {
    ```
    ```

```
printf("\nHello,, I am child! Now I'll run date\n*);
execlp("/bin/date", "/bin/date", (char*)0; /* date라는 새로운 프로그램 */
printf("\n Hello, I am Parent!\n");

}
```

- exec() 만난 이후 date라는 새로운 프로그램으로 덮어짐
- exec() 이후 코드는 실행 되지 못함. (print문 실행 x)
- exec() 코드 프로그램 2번 적어주고, 전달할 argument 들을 적어준다. " "안에 적고 ,로 구분하고 (char*)0 작성
  - 그냥 exec()가 어떻게 구성되는 지 보여주신 것.(몰라도 됨)

wait()

- 커널은 child가 종료될 때 까지 프로세스를 sleep시킨다(block 상태)
- child Process가 종료되면 커널은 프로세스를 깨운다. (ready)
- ```
  main(
      let ChildPID;
      S1
      ChildPid = fork()
      if (ChildPid === 0)
      {/* code for child/*}
      else{
      wait()
  }
  S2
  )
  ```
- 터미널에서 명령 실행했을 때

exit()

- 프로그램 종료할 때 시스템 콜

```
main(
    let ChildPID;
    S1
    exit()
    ChildPid = fork()
    if (ChildPid === 0)
    {/* code for child/*}
    else{
    wait()
}
S2
)
```

- 명시적으로 사용될 수도 있고 끝에 자동으로 호출 될 수도 있음
- 비자발적 종료
  - 부모 프로세스가 자식 강제 종료
  - 키보드로 kill, break
```

    - 부모가 종료되는 경우

프로세스 간 협력

- 독립적 프로세스
    - 원칙적. 각자 주소 공간 가지고 실행
- 협력 프로세스
- 프로세스 간 협력 메커니즘(IPC : InterProcess Communication)
    - message passing: `커널` 통해서 메시지 전달
        - 프로세스 사이에 공유 변수를 일체 사용하지 않음
        - Direct Communication: 통신하려는 프로세스 이름 명시적으로 표시
        - Indirect Communication: mailbox(또는 port) 통해 메시지 간접 전달
    - shared memory:
        - 서로 다른 프로세스 간 일부 주소 공간 공유
        - 커널에 shared memory 쓴다는 시스템 콜 필요
    - thread
        - 사실 상 하나의 프로세스 (프로세스 간 협력은 아님)
        - process 구성하는 thread간 주소 공간을 공유