

이화여자대학교

EWHA WOMANS UNIVERSITY

프로세스 생성 (Process Creation)

- 부모 프로세스(Parent process)가 자식 프로세스(children process) 생성
- 프로세스의 트리(계층 구조) 형성
 - 1. 주소공간.
 - 2. Counter.(패러미터 전달) 복제함.
- 프로세스는 자원을 필요로 함
 - 운영체제로부터 받는다
 - 부모와 공유한다
- 자원의 공유
 - 부모와 자식이 모든 자원을 공유하는 모델. Code, Data, Stack. 메모리 동일여부.
 - 일부를 공유하는 모델 → advance된 개념.
 - 전혀 공유하지 않는 모델 → 완전한, 별다른 CPU, 메모리 경쟁.
- 수행 (Execution)
 - 부모와 자식은 공존하며 수행되는 모델
 - 자식이 종료(terminate)될 때까지 부모가 기다리는(wait) 모델

Blocked 상태

0:00:33

-10:03 1x 1080p

부모 프로세스 1개가 여러 자식 프로세스 생성.

(Copy on Write (COW) 기법.
Write가 발생시 새로운 것을 만들었다.)

이화여자대학교

EWHA WOMANS UNIVERSITY

프로세스 생성 (Process Creation)

- 주소 공간 (Address space) (Code, Data, Stack)
 - 자식은 부모의 공간을 복사함 (binary and OS data) (현재까지 데이터도 복제)
 - 자식은 그 공간에 새로운 프로그램을 올림
- 유닉스의 예
 - fork() 시스템 콜이 새로운 프로세스를 생성 → 즉, 운영체제가 자원 할당.
 - 부모를 그대로 복사 (OS data except PID + binary)
 - 주소 공간 할당
 - fork 다음에 이어지는 exec() 시스템 콜을 통해 새로운 프로그램은 메모리에 올림 → 새로운 프로그램.

0:03:24

-7:12 1x 1080p

프로세스의 생성은 보통 2단계를 거친다.

1. 부모 프로세스를 복제하는 단계) 독립적이기
2. 새로운 프로그램을 덮어씌우는 단계) 안 수행할 수도 있음

이화여자대학교

EWHA WOMANS UNIVERSITY

프로세스 종료 (Process Termination)

- 프로세스가 마지막 명령을 수행한 후 운영체제에게 이를 알려줌 (exit)
 - 자식이 부모에게 output data를 보냄 (via wait). 자식이 먼저 죽음.
 - 프로세스의 각종 자원들이 운영체제에게 반납됨 wait로 데이터 전송.
- 부모 프로세스가 자식의 수행을 종료시킴 (abort)
 - 자식이 할당 자원의 한계치를 넘어섬 → 강제 종료.
 - 자식에게 할당된 리소스가 더 이상 필요하지 않음
 - 부모가 종료(exit)하는 경우
 - 운영체제는 부모 프로세스가 종료하는 경우 자식이 더 이상 수행되도록 두지 않는다
 - 단계적인 종료 자식이 먼저 죽어야 하기 때문에 (부모가 wait까지 단계적으로 죽인다.

0:06:37

-3:59 1x 1080p

exit: 프로그램 종로 시켜줌. (C언어의 마지막 증괄호를 컴파일할 때 exit 생성.)

이화여자대학교

EWHA WOMANS UNIVERSITY

fork() 시스템 콜

- A process is created by the fork() system call.
 - creates a new address space that is a duplicate of the caller.

```

int main()
{
    int pid;
    pid = fork();
    if (pid == 0) /* this is child */
        printf("Hello, I am child\n");
    else if (pid > 0) /* this is parent */
        printf("Hello, I am parent\n");
}

```

Parent process
pid > 0

Child process
pid = 0

Child.

이 아휘의 서약. (부모 프로세스 원형 그대로 복제. 따라서 부모 프로세스의 리소스가 fork 실행됨을 가리키는 값이므로 저 부분부터 시작하게 된다!)

0:08:17

fork() 생성시 생각할 수 있는 두가지 문제. fork return 값

1. 누가 parents고 누가 children 인가. → 부모 프로세스는 자식의 pid 리턴. (자식 pid)
2. 동시작업 수행만 하는 것 아냐? 자식 프로세스는 이를 거른다.

↳ fork return 값이 상이하므로
다음을 통해 다른 작업수행가능.

이화여자대학교
EWHA WOMANS UNIVERSITY

exec() 시스템 콜

- A process can execute a different program by the `exec()` system call.
 - replaces the memory image of the caller with a new program.

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

/* this is child */
→ 새로운 프로그램으로 대체된다.
(32비트 시스템, 32비트 argument, 64비트)
execl 시작시 main부터 시작
다시 어떤 프로그램으로 돌아올 수 없음

`exec|p`를 꼭 자식 프로세스에서 실행시키지 않아도 됨.

```
int main() {
    printf( );
    execlp( );
    printf( );
}
```

이 부분은 실행 안됨. `execlp`이 의해
서 프로그램이 되었으므로.

이화여자대학교
EWHA WOMANS UNIVERSITY

wait() 시스템 콜

- 프로세스 A가 `wait()` 시스템 콜을 호출하면
 - 커널은 child가 종료될 때까지 프로세스 A를 sleep시킨다 (block 상태)
 - Child process가 종료되면 커널은 프로세스 A를 깨운다 (ready 상태)

```
main (
{
    int childPID;
    S;
    childPID = fork();
    if (childPID == 0)
        /* code for child process */
    else {
        wait();
    }
    S;
}
```

fork()

Parent: Code, Data, Stack
Child: Code, Data, Stack
childPID = 0
childPID = XXX

프로세스를 blocked 상태로 전환하는 것.

보통 자식 프로세스를 만든 다음에 `wait` 시스템 콜을 함.
자식 프로세스가 종료되기를 기다리면서 blocked 상태가 됨.

종료되면 blocked → ready 상태가 된다.

↳ Shell이 Program을 실행시키면 종료시까지
Shell 입력불가한 것이 `wait` system call 모델을
따라한 것.

Shell 부모 프로세스가 Program 자식 프로세스 실행 후
종료시까지 가만히 있어야 함.

이화여자대학교
EWHA WOMANS UNIVERSITY

exit() 시스템 콜

- 프로세스의 종료
 - 자발적 종료
 - 마지막 statement 수행 후 `exit()` 시스템 콜을 통해
 - 프로그램에 명시적으로 적어주지 않아도 main 함수가 리턴되는 위치
에 컴파일러가 넣어줌
 - 비자발적 종료
 - 부모 프로세스가 자식 프로세스를 강제 종료시킴
 - 자식 프로세스가 한계치를 넘어서는 자원 요청
 - 자식에게 할당된 태스크가 더 이상 필요하지 않음
 - 키보드로 `kill`, `break` 등을 친 경우 사람이 원하는 것
 - 부모가 종료하는 경우
 - 부모 프로세스가 종료하기 전에 자식들이 먼저 종료됨

1. 명시적 exit() 시스템 콜.

```
int main()
{
    int pid;
    printf("Hello, I am child!\n");
    exit();
    pid = fork();
    if (pid == 0) /* this is child */
        printf("Hello, I am child!\n");
    else if (pid > 0) /* this is parent */
        printf("Hello, I am parent!\n");
}
```

프로세스와 관련된 시스템 콜.

`fork()` creat a child (copy)

`exec()` overlay new image.

`wait()` sleep until child is done

`exit()` frees all the resources, notify parent.

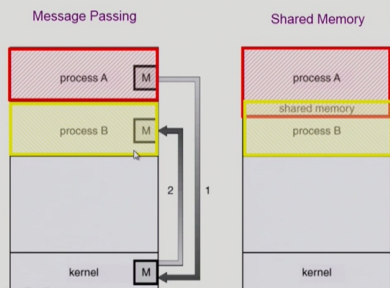
프로세스 간 협력

- ➔ 독립적 프로세스(Independent process)
 - ✓ 프로세스는 각자의 주소 공간을 가지고 수행되므로 원칙적으로 하나의 프로세스는 다른 프로세스의 수행에 영향을 미치지 못함
 - ➔ 협력 프로세스(Cooperating process)
 - ✓ 프로세스 협력 메커니즘을 통해 하나의 프로세스가 다른 프로세스의 수행에 영향을 미칠 수 있음
 - ➔ 프로세스 간 협력 메커니즘(IPC: Interprocess Communication)
 - ✓ 메시지를 전달하는 방법
 - **message passing**: 커널을 통해 메시지 전달 **프로세스는 직접 전달 X**
 - ✓ 주소 공간을 공유하는 방법
 - **shared memory**: 서로 다른 프로세스 간에도 일부 주소 공간을 공유하게 함은 shared memory 메커니즘이 있음
- ★ **thread**: thread는 사실상 하나의 프로세스이므로 프로세스 간 협력으로 보기는 어렵지만 동일한 process를 구성하는 thread들 간에는 주소 공간을 공유하므로 협력이 가능

Message Passing

- 항상 커널을 통해 메시지 전달 가능. 인터페이스 방식이 사용됨.**
- ➔ Message system
 - ✓ 프로세스 사이에 공유 변수(shared variable)를 일체 사용하지 않고 통신하는 시스템
 - ➔ Direct Communication
 - ✓ 통신하려는 프로세스의 이름을 명시적으로 표시
- Process P → Process Q
Send (Q, message) Receive (P, message)
- ➔ Indirect Communication
 - ✓ mailbox (또는 port)를 통해 메시지를 간접 전달
- Process P → Mailbox M → Process Q
Send (M, message) Receive (M, message)
- 당연히 커널이 관여**

Interprocess Communication



Shared Memory도 프로세스가 직접 접근 못하고 kernel을 통해 시스템 콜로 맵핑 가능.

