

프로세스 생성(Process Creation)

- 부모 프로세스(Parent Process)가 자식 프로세스(Children process) 생성

- 프로세스의 태(계층구조) 형성

- 프로세스는 자원을 필요로 함

- ↳ 운영체제로 부터 받는다.
- ↳ 부모와 공유 한다.

- 자원의 공유

- ↳ 부모와 자식이 모든 자원을 공유하는 모델
- ↳ 일부를 공유하는 모델
- ↳ 전혀 공유하지 않는 모델

- 수행(Execution)

- ↳ 부모와 자식은 공존하며 수행되는 모델
- ↳ 자식이 종료(terminate)될 때까지 부모가 기다리는 (wait) 모델
= Blocked

- 주소공간(Address Space)

- ↳ 자식은 부모의 공간을 복사함 (binary and OS data)
- ↳ 자식은 그 공간에 새로운 프로세스를 물림.

- 유닉스의 예

- ↳ fork() 시스템 콜이 새로운 프로세스를 생성
 - 부모를 그대로 복사(OS data except PID + binary)
 - 주소 공간 할당
- ↳ fork 다음에 이어지는 exec() 시스템 콜을 통해 새로운 프로그램을 메모리에 풀림.

↳ 대부분 운영체제에서는 COW가 그렇지 않아 나쁜 운영체제, 메모리 관리, 파일시스템 같은 문제에서 Copy-On-Write 가 자주 나온다.

↳ 리소스 같은 풀 더 효율적인 운영체제에서는 접근 가능하지 않고
자식의 부모의 주소공간을 공유하고 있음 (Program Counter 만 한계(Cop)해서 똑같은 위치를 가리키고 있음)
그러다가 변수의 data 값이 달라 진다던가, 딸라인 함수호출, 스택에 쌓인 내용들이 달라지면서
각자의 값을 가지게되면 그때 공유하던 부모의 메모리 공간 일부를 깨끗해서 자유롭게 갖게됨.

→ 경우에 따라서 부모와 자식이 자원을

공유하는 경우도 있고 그렇지 않은 경우도 있다.

운영체제로는 자원을 공유하지 않는다.

부모 프로세스가 자식 프로세스를 생성하면 서로 빙드의
프로세스 이기 때문에 각자 CPU를 얻으려고 경쟁하고,
메모리를 더 많이 얻으려고 경쟁한다.

↳ 복제 후, 덮어 씁.

→ 부모 프로세스가 자식 프로세스를 직접 생성한다고 말했지만
운영체제를 통해서만 생성이 가능. ↳ fork(), exec() 같은 System call
이용해 운영체제에 부탁.

프로세스 종료 (Process Termination)

- 프로세스가 마지막 명령을 수행한 후 운영체계에게 이를 알리음 (exit)
 - ↳ 자발적 종료
- ↳ 자식이 부모에게 output data를 보냄. (via wait) \Rightarrow 항상 자식 프로세스 먼저 주고 그 다음 부모 프로세스
- ↳ 프로세스의 각종 자원들이 운영체계에게 반납됨.
- 부모 프로세스가 자식의 수행을 종료시킴. (abort)
 - ↳ 자식이 할당 자원의 한계를 넘어서 비 자발적 종료
 - ↳ 자식에게 할당된 태스크가 더 이상 필요하지 않음
 - ↳ 부모가 종료 (exit) 하는 경우
 - 운영체제는 부모 프로세스 종료하는 경우 자식이 더 이상 수행되도록 하기 않는다.
 - 단계적인 종료

fork() 시스템 콜

- A process is created by the fork() system call.

↳ Create a new address space that is a duplicate of the caller.

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

X fork()를 통해 process를 생성했을 때 생길 수 있는 문제점

① 같은 부모로부터 실행하기 때문에 자식 프로세스가 부모 프로세스를 통해资源共享할 수 있음.

② 부모와 똑같이 만들어 놓기 때문에 세상에 있는 프로그램은 모두 똑같은 제작 흐름을 따라야 할 수 있음.

↳ 이를 해결하기 위해 부모와 자식 프로세스의 결과 값 (return Value)를 다르게 함.

부모 프로세스는 fork의 결과값 (PID = 자식 프로세스의 번호)을 '양수'로 인지되고,

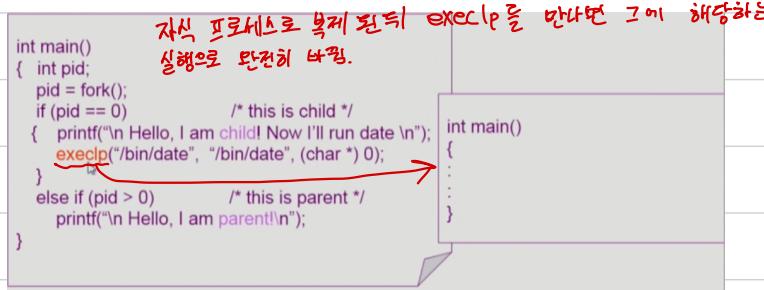
자식 프로세스는 fork를 한 결과 값인 '0'을 인함. → 이를 통해 부모 자식 구분 가능.

결과 값 (PID)을 통해 부모와 자식 간에 다른 일을 시킬 수 있음.

exec() 시스템 콜

- A process can execute a different program by the exec() system call.

v replace the memory image of the caller with a new program.



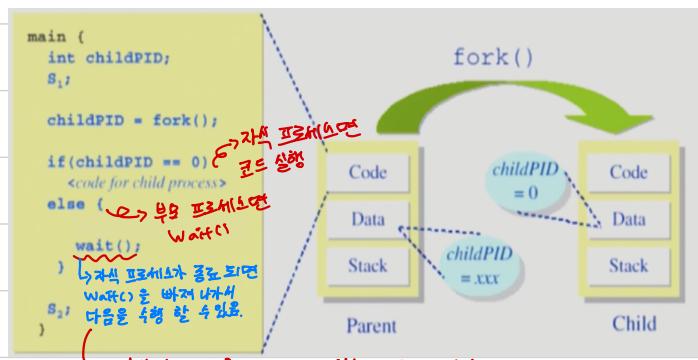
Wait() 시스템 콜

- 프로세스 A가 wait() 시스템 콜을 호출하면

v 커널은 child가 종료될 때까지 프로세스 A를 sleep 시킨다. (block 상태)

v Child process가 종료되면 커널은 프로세스 A를 깨운다. (ready 상태)

오래 걸리는 event를 기다리고
event가 만족되면 다시 CPU를 얻을 수 있는
ready 상태로 돌아오는 것



* 대표적인 예)

LINUX terminal에서 명령 입력 후 엔터를 누르면 명령이 실행되고

그 명령이 끝날 때까지 terminal이 커서가 없다가 명령이 끝나면 커서가 다음 줄에 깔빡이고 있음.

exit() 시스템 콜

- 프로세스의 종료

▼ 자발적 종료

- 마지막 statement 수행 후 exit() 시스템 콜을 통해
- 프로그램이 명시적으로 작업하지 않아도 main 함수가 return 되는 경우에
컴파일러가 넣기를

▼ 비 자발적 종료

- 부모 프로세스가 자식 프로세스를 함께 종료시킴 \rightarrow 부모 프로세스
- 자식 프로세스가 한 처리를 끝에서는 자립 요청
- 자식에게 할당된 메모리 더 이상 필요하지 않을 때
- 키보드로 Kill, break 등을 친 경우 \rightarrow 사용자
- 부모가 종료하는 경우
- 부모 프로세스가 종료하기 전에 자식들이 먼저 종료됨

프로세스와 관련한 시스템 콜

- fork() Create 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

- ✓ 프로세스는 각자의 주소 공간을 가지고 수행 되므로 원칙적으로 하나의 프로세스는 다른 프로세스의 수행에 영향을 미치지 못함

- 협력 프로세스

- ✓ 프로세스 협력 메커니즘을 통해 하나의 프로세스가 다른 프로세스의 수행에 영향을 미칠 수 있음.

↳ 프로세스 간 정보를 주고 받을 수 있는 방법

- 프로세스 간 협력 메커니즘 (IPC: Interprocess Communication)

- ✓ 메시지를 전달하는 방법

message passing: 커널을 통해 메시지 전달

- ✓ 주소 공간을 공유하는 방법

shared memory: 서로 다른 프로세스 간에도 일부 주소 공간을 공유하게 하는 shared memory 메커니즘이 있음

* thread: thread는 사실상 하나의 프로세스 이므로 프로세스간 협력으로 보기기는 어렵지만 동일한 Process를 구성하는 thread들 간에는 주소 공간을 공유 하므로 협력이 가능

Message Passing

- Message system

- ✓ 프로세스 사이에 공유변수 (shared variable)를 설정해 사용하지 않고 통신하는 시스템.

- Direct Communication

- ✓ 통신하려는 프로세스의 이름을 명시적으로 표시



→ 커널을 통해 전달한다는 점은 같다.

message를 받아 놓은 프로세스를 명시 하느냐 아니면 Mail box에 message를 끌어 놓느냐 두 가지 방법의 차이

- Indirect Communication

- ✓ mail box (또는 port)를 통해 메시지를 간접 전달



→ 아무나 꺼내가라 할 수로 있음.

Interprocess Communication

