**CSED211 Homework5**

20210774 김주은

**1. Exercise 8.16 on page 826.**

int counter = 1; 로 선언된 counter라는 전역변수가 있다.

main function에서, if 조건문을 거치며 fork() 함수를 호출하게 되고, child가 생긴다.

child가 생기므로, fork() != 0 에 의해 else 구문을 실행한다.

이때, counter++;을 통해 counter 값이 1만큼 증가하므로 counter 값이 2가 된다.

printf(“counter = %d\n”, counter); 에서 counter 값이 2이므로, counter = 2라고 출력된다.

Answer: counter = 2

**2. Exercise 8.17 on page 827**

Enumerate all of the possible outputs of the program in Practice Problem 8.4.

텍스트이(가) 표시된 사진

자동 생성된 설명

Fork 함수는 자식에서는 0을 return하고, 부모는 자식의 PID를 return한다.

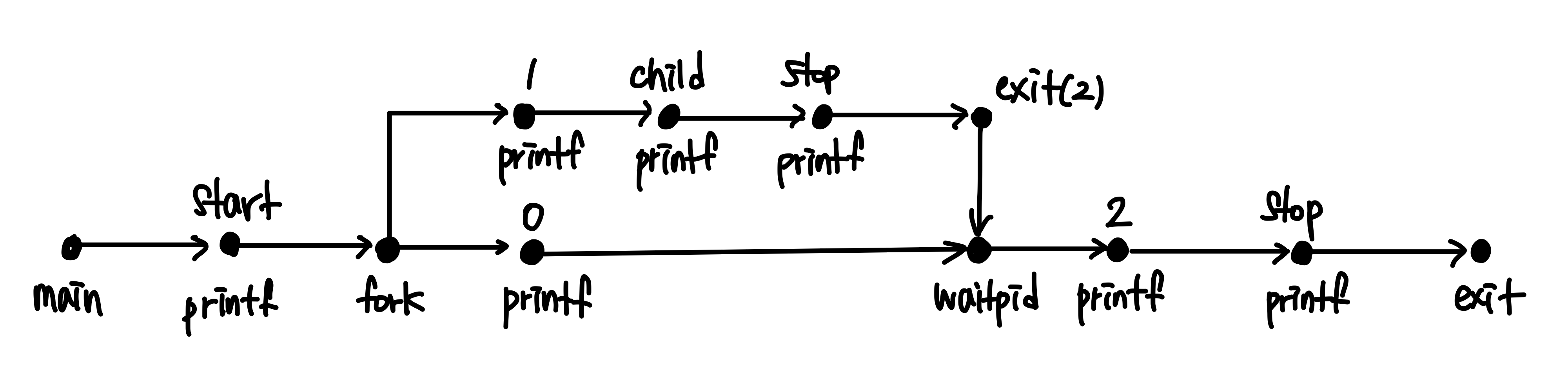
위의 if-else if 조건문에서 pid == 0의 비교문을 통해서,

자식의 경우는 Fork함수가 0을 return하므로 pid == 0이 true이고, printf(“Child\n”);을 수행한다.

부모의 경우는 Fork함수가 자식의 PID를 return하므로, else if 구문을 실행할 것이다.

하지만, 부모와 자식의 경우 실행되는 순서가 바뀔 수 있기 때문에 여러 가지의 possible output이 생긴다.

이를 바탕으로 process graph를 그리면 다음과 같다.



Start는 fork()전에 출력되므로 항상 제일 먼저 출력되어야 한다.

이후 fork()가 호출된 이후에 자식이 생성되면 자식은 알아서 수행이 되고, 부모의 경우는 waitpid함수에 의해 자식이 종료될 때까지 기다리게 된다.

그래서 결국 자식이 종료된 이후에 WEXITSTATUS(status)의 리턴값을 출력하므로 여기에 대해서도 순서가 존재하는 것이다.

결국 순서가 바뀔 수 있는 것은 위에서 병렬적으로 그려져 있는 부분만 해당된다. 그러므로, 가능한 출력의 경우에는 총 4가지다.

Answer: 총 4가지

1)

Start

0

1

Child

Stop

2

Stop

2)

Start

1

0

Child

Stop

2

Stop

3)

Start

1

Child

0

Stop

2

Stop

4)

Start

1

Child

Stop

0

2

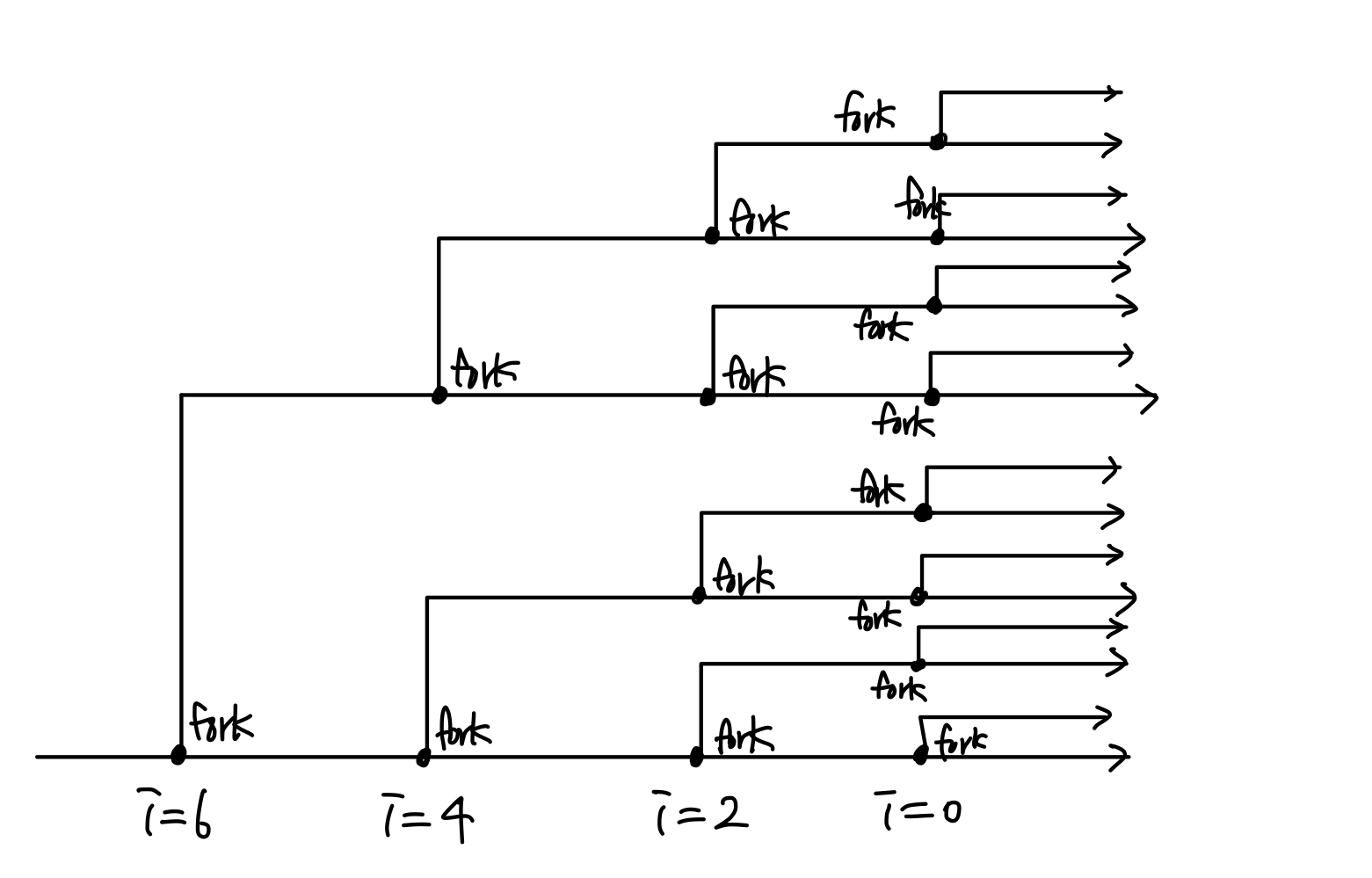
Stop

**3. Exercise 8.19 on page 828 with n = 7**

n=7이면

for ( i = 6 ; i >= 0 ; i -= 2) 이므로, I = 6, 4, 2, 0일때 수행된다.

이를 바탕으로 자식을 생성하는 것을 생각했을 때 process graph는 다음과 같다.



총 2^4 개의 line의 “hello”라는 output이 나올 것이다.

Answer: 16

**4. Exercise 8.21 on page 828**

fork()를 호출한 후 자식이 생성되고, 부모와 랜덤한 순서로 수행되면서 여러 가능한 output이 생긴다. 그러나 fork() 호출 전 printf(“p”);가 먼저 나오므로 p는 항상 먼저 출력될 것이다.

fork() 함수는 자식일 경우는 0을 return하고, 부모인 경우는 자식의 PID를 return하기 때문에 자식과 부모의 경우에 따라 q를 출력하거나 r 을 출력할 것이다. 그리고 이들의 출력 순서는 바뀔 수 있다. 그러므로 pqr, prq가 가능한 출력이 된다.

Answer : pqr, prq

5. Exercise 9.11 on page 912

A. Virtual address format

Virtual address : 0x027c이고, 이를 2진수로 변환하면 0000001001111100 이므로, 여기서 하위 14bit를 가져와 총 14bit의 형태로 표시하면 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

B. Address translation

각 페이지의 크기가 2^6 = 64 바이트이므로 하위 6비트를 제외한 상위 8비트가 VPN이 된다.

그러므로, VPN 은 이진수로 00001001의 값을 가지므로 0x09이다.

VPN에서 하위 2비트는 TLBI로 사용되고 상위 비트들은 TLBT로 사용된다.

그러므로 TLB index는 1, TLB tag는 2이다.

하지만, index가 1인 set에서 tag가 2인 page는 valid=0이므로, TLB miss가 발생한다.

하지만, page table에서 VPN이 09인 곳에 PPN값이 0x17이고, Valid값이 1임을 통해 page fault는 발생하지 않는다는 것을 알 수 있다.

|  |  |
| --- | --- |
| Parameter | Value |
| VPN | 0x09 |
| TLB index | 0x01 |
| TLB tag | 0x02 |
| TLB hit? (Y/N) | N |
| Page fault? (Y/N) | N |
| PPN | 0x17 |

C. Physical address format

PPO와 VPO는 같으므로 virtual address에서의 하위 6bits가 physical address에서의 하위 6bit와 일치한다. (111100)

상위 6bit는 PPN에 따라 0x17의 값이 된다. 그러므로 physical address는 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

D. Physical memory reference

Byte offset인 CO는 physical memory의 하위 2bits이므로, 0이다.

Carry index는 그 다음 하위 4bit이므로, 1111이고, 15다.

Carry tag는 나머지 상위 bit들이므로 010111이고 0x17이다.

Cache를 보게 되면 index F에서 tag가 14이기 때문에 17과 일치하지 않으므로 cache miss가 일어나고, return되는 cache byte도 없다.

|  |  |
| --- | --- |
| Parameter | Value |
| Byte offset | 0x00 |
| Cache index | 0x0F |
| Cache tag | 0x17 |
| Cache hit? (Y/N) | N |
| Cache byte returned | - |

6. Exercise 9.14 on page 914

- C 파일에 코드 첨부