1. Zombie process

1-1. Process control block이 지워지지 않고 계속 남아 있게되어 memory leak이 발생하게 된다. 궁극적으로 out of memory가 발생할 수 있다.

1-2. signal 과 같은 IPC를 위한 값들을 저장해 놓을 수가 없게 된다. child process가 exit(v); 를 하면 v 값이 parent process에게 전달되어야 하는데 이런 값을 전달할 방법이 없어진다.

2-1. IBM PC page table

- 20 bit address line -> 물리주소는 2^{20} (1MB)까지 가능하지만 256KB (18bit만 사용)
- 최대 프로그램 6MB, 디스크 크기가 10MB
  - 가상주소공간를 8MB(2^{23})로 잡으면 가상주소를 위해서 23bit 가 필요하다.
  - 페이지 크기가 512 byte -> page offset 9 bit.
  - 가상 페이지 # : 0 ~ $2^{14} - 1$
  - 물리 페이지 # : 0 ~ $2^{9} - 1$
  - PTE : 물리페이지#(9bit) , valid, dirty, R/W등 총16bit
  - 하나의 page(512B)안에 256개의 PTE

2-1. IBM PC page table

- 1 level이면 2^{14} 개의 PTE필요
  - 2^{14} / 2^{8} = 64 page(32KB) 필요
  - 메모리가 256KB이므로 8개의 프로세스가 있다면 끝
- 2 level이라면 6+8 로 나누어서 root page에 있는 64개의 entry가 2nd level page를 가리키게 하면 그 중에서 실제 사용되는 것만 할당할 수 있게 해서 작은 프로세스가 사용하는 page table의 양을 줄일 수 있다.
  - 256KB를 사용하는 프로세스라면 leaf page 두개와 root page 하나, 3페이지만 사용하면 된다.
  - 물론 이때 128KB씩은 연속되어 사용되어야 한다.

2-2. IBM PC page table

- 1MB를 사용하는 프로세스라면 leaf page 8개와 root page 하나, 9페이지만 사용하면 된다.
  - 1개의 page table에는 256 entry
    - 256 * 512B = 128KB
  - 하나의 page table page가 128KB를 커버한다.
    - 128KB * 8 = 1MB

3. TLB

- Code segment : 2KB
- Global data segment : 4KB
- Local data segment : 3KB
- 전체 메모리 양 : 9KB, 18 page

- TLB가 18개가 있으면 항상 TLB hit 일어난다.
- TLB가 하나라면 code/data 에서 계속 miss 발생
- TLB가 2(1 code+1 data)개만 있어도 최소한으로는 돌아간다.
- TLB가 3(1 code+1 global data + 1 local data)개면 조금 더 잘 돌아갈 수 있다.

4. Signal vs interrupt

- Interrupt가 발생하면 그 순간에 돌아가던 instruction이 끝나는 대로 바로 처리가 된다.
- 하지만 signal은 발생후 system call/context switch가 발생해서 다시 그 프로세스로 돌아오기 전에 확인하고 처리하게 된다.
- 만일 진행중인 프로세스가 system call이나 다른 interrupt가 발생하지 않는다면 오랫동안 기다리게 될 수도 있다.

5. fork()

- Child process를 위한 pid 가 하나 새로 만들어지고 만들어진 child의 Process Control Block은 parent 의 PCB를 복사해서 만들어진다.
- Address space도 그대로 copy가 된다.
- 이 모든 것이 copy-on-write가 되기 때문에 child가 값을 바꾸면 원래 copy는 그대로 있고 새로운 copy가 만들어져서 고쳐진다.

6. Call vs return()

- 일반 함수는 한번 불리고 한번 돌아온다.
- fork()는 한번 불리지만 parent와 child로 한번씩 총 두 번 return 된다.
- longjmp는 setjmp 해놓은 위치로 돌아가는데 여러 번 call이 되고 여러 번 같은 위치로 돌아갈 수 있다.

7. pipe()

- .

```
int fd[2];
if (pipe(fd) != 0) {
printf ("Error creating pipe. %s", strerror(errno));
exit(errno); }
If (fork()) /* child1 */
   { dup2(fd[1],1); execve("ls-l"); }
else        /* child2 */
   { dup2(fd(0),0); execve("sort") }
```

8. Relocation entry

- Offset : relocation을 해야 하는 곳의 주소
- Symbol : reference 해야 하는 위치
  - 경우에 따라서 단순symbol이 아니라 symbol+value(array index * size) 인 경우도 있다.
- Type : relocation type
- Value : PC-relative인 경우 그 위치에 있을때의 PC 보정값 (PC = offset - value)

- Symbol table에서 symbol을 찾으면 그 위치(주소)를 알게되고 absolute냐 PC-relative냐에 따라서 offset 위치에 값을 바꿔준다.