# Punteros y direcciones Vs fork()

```
int p = 14, pp = 10;
if (fork() == 0) {
        printf("hijo: direccion de p = [%p] valor = %d\n",&p,p);
        printf("hijo: direccion de pp = [%p] valor = %d\n",&pp,pp);
        p = 20;
        printf("hijo: direccion de p = [%p] valor = %d\n",&p,p);
        printf("hijo: direccion de pp = [%p] valor = %d\n",&pp,pp);
        printf("hijo: fin!\n”);
        exit(0);
} else {
        printf("padre: direccion de p = [%p] valor = %d\n",&p,p);
        printf("padre: direccion de pp = [%p] valor = %d\n",&pp,pp);
        pp = 20;
        printf("padre: direccion de pp = [%p] valor = %d\n",&pp,pp);
        wait0);
        printf("padre: hijo finalizado!\n");
        printf("padre: direccion de p = [%p] valor = %d\n",&p,p);
        printf("padre: direccion de pp = [%p] valor = %d\n",&pp,pp);
}
```

Tal como indica la teoría, p y pp son variables independientes. Existe “p” en el padre y existe otra “p” en el hijo. Lo mismo sucede con pp.
%p nos muestra la dirección lógica de la variable y en Linux vemos que la dirección no cambia, no obstante, el comportamiento es acorde a la teoría.

# Espacio de direcciones virtuales

- Cada proceso mapea direcciones virtuales a direcciones físicas de memoria
- Luego de fork() estas direcciones virtuales se marcan como “read only”
- Si un proceso hijo intenta cambiar una variable → se asigna una nueva pagina de memoria física no “read only”
- La dirección virtual es la misma, pero esta apuntando a una ubicación distinta de la memoria física

Proceso COW (copy on write)