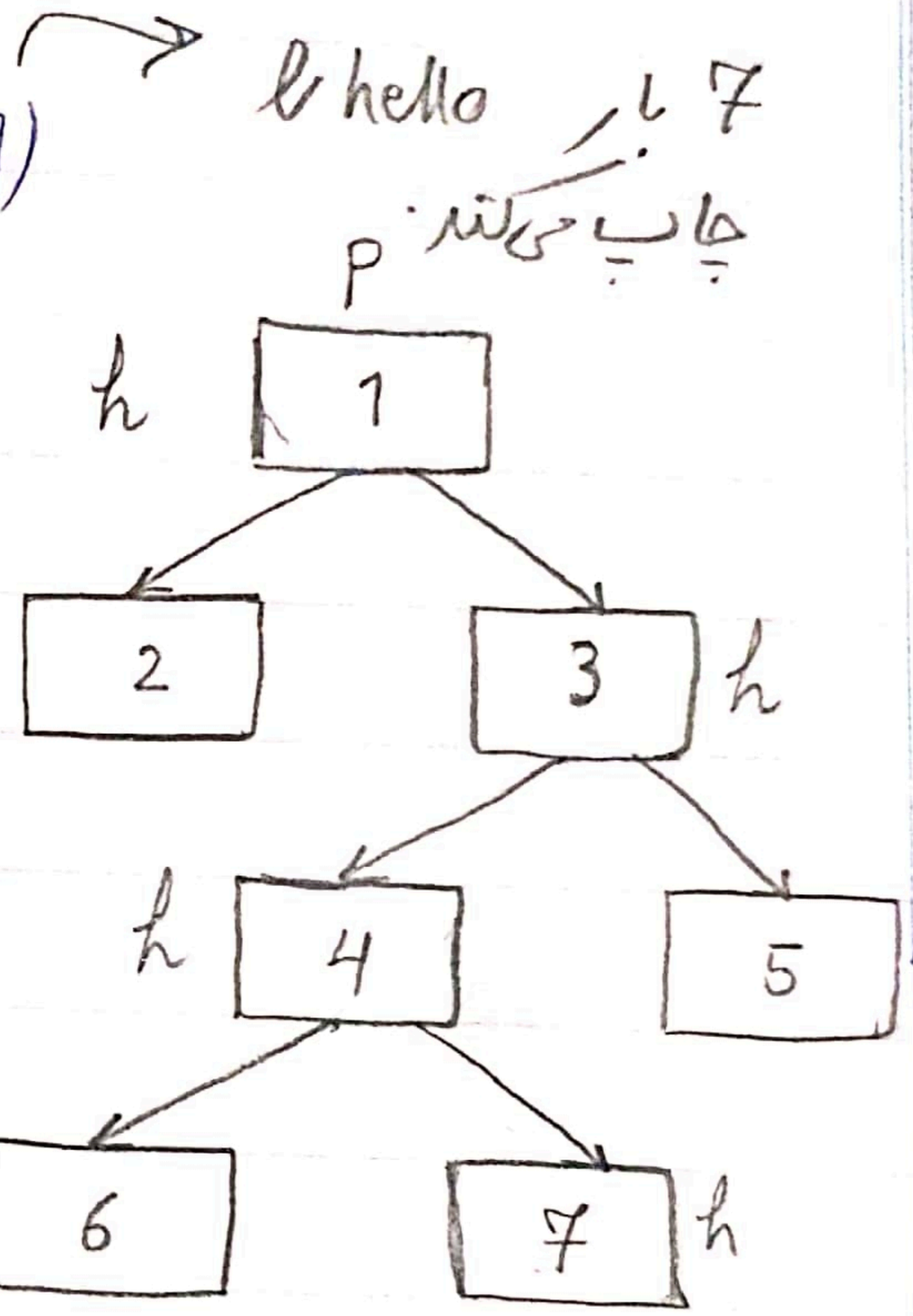
کار کلاسی جلسه دوم

```
int main (int argc , char * argv[])
{
1.    if (fork() && (!fork()))
2.    {
3.        if( fork() || fork())
4.        {
5.            fork();
6.        }
7.    }
8.    printf( "hello\n");
      return 0;
}
```

hello ۷ بار چاپ می‌کند.



zero ، negative ، positive به سه صورت fork() می‌تواند خروجی می‌باشد. زمانی که processor ، تواند child process را انجام دهد، fork() ، مقدار منفی برمی‌گرداند. همچنین مقدار positive ، return پدر سرورهای parent می‌باشد که به مکان پراسور child اشاره می‌کند. return پراسورهای child هم مقدار صفر می‌باشد.

در ابتدا در خط 6، زمانی که وارد حلقه if شدیم به fork رسید، یک child تولید می‌کند. اگر P_2 را به عنوان child در نظر بگیریم، قسمت اول بخش and، غلط می‌شود، چرا که P_2 چون child است، مقدار صفر را برمی‌گرداند و هرچیزی با صفر، صفر می‌شود. پس داخل if نمی‌شود. اکنون P_1 را به عنوان parent در نظر می‌گیریم. چون P_1، parent است، پس حرفی مخالف صفر و عددی مثبت برمی‌گرداند و بخش اول if، درست می‌شود. پس وقتی به fork ای که در بخش دوم and قرار دارد، رسید، یک child دیگر به نام P_3 تولید می‌شود. در اینجا اگر P_3 را به عنوان child در نظر بگیریم، حرفی برای با صفر خواهد داشت که باعث می‌شود if اول درست نشود و وارد if دوم می‌شویم. در if دوم زمانی که به fork اول رسیدیم، یک child به اسم P_4 درست می‌شود. حال اگر P_3 را به عنوان child در نظر بگیریم، بخش اول if دوم نادرست می‌شود در زمانی که به if دوم (بخش دوم) می‌رسد، یک child دیگر به اسم P_5 به وجود می‌آید که اگر P_5 را به عنوان child در نظر بگیریم، شرط درست بودن if دوم نقض می‌شود و دیگر وارد if دوم نمی‌شود. اکنون اگر P_4 را به عنوان child در نظر بگیریم، قسمت اول if غلط می‌شود، پس ادامه می‌دهد تا به بخش دوم برسد. آن‌زمانی که به fork دوم رسید، یک child به عنوان P_6 تشکیل می‌دهد که چون P_6 هم مقدار صفر را برمی‌گرداند، دیگر داخل if نمی‌شود. حال اگر P_4 را به عنوان Parent در نظر بگیریم، بخش اول if درست می‌شود و چون or است، نیازی به ادامه دادن برای بررسی درست بودن if نداریم. بنابراین وارد if دوم شده و به fork که می‌رسد، child P_7 را تشکیل می‌دهد.