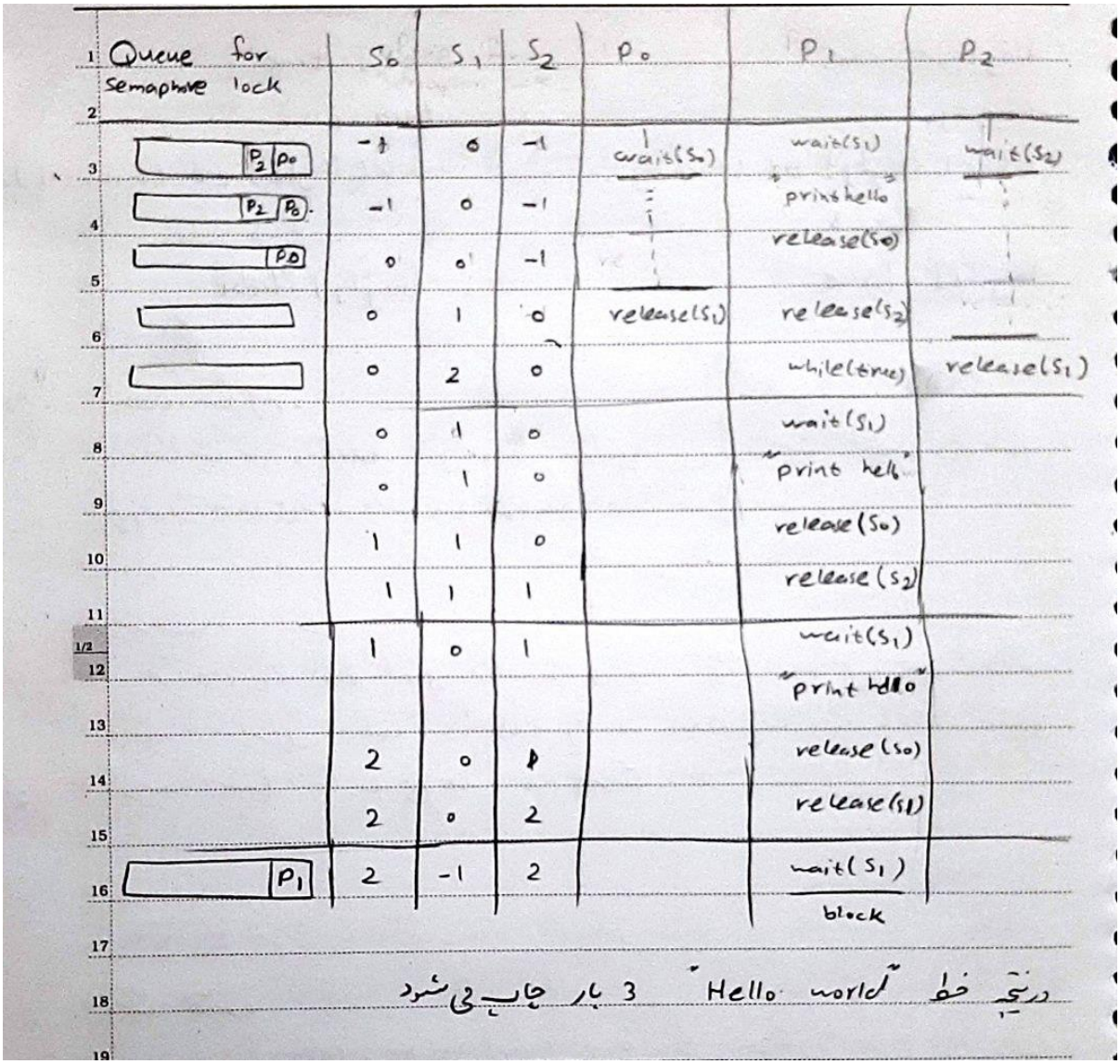
سوال اول:

از آنجا که فقط همین دو قسمت که درون compare_and_swap اند به available دسترسی دارند، دستورات acquire, release اتمیک اجرا می شوند.

```
bool lock = flase;
bool available = false;
void acquire(){
   while(!compare_and_swap(&lock, 0, 1));
   while (!available);
   available = false;
   lock = false;
}
void release(){
   while(!compare_and_swap(&lock, 0, 1));
   available = true;
   lock = false;
}
```

سوال دوم:



سوال سوم:

الف)

```
int compare_and_swap(int *value, int expected, int new_value) {
   int temp = *value;
   if (*value == expected)
      *value = new_value;
   return temp; /* old value */
}

boolean test_and_set(boolean *target) {
   return compare_and_swap(target, *target, 1)
}
```

ب) خیر زیرا ما به اتمیک بودن آن ناحیه ای که قرار است از این دستورات استفاده کنیم نیاز داریم و بقیه دستورات اگر بین شان context switch شود برای مان مشکلی پیش نمی آورند که اگر مشکلی پیش بیاید باید از دستورات Compare_and_Swap وTest_and_Set برای اتمیک اجرا شدن آنها استفاده کنیم.

ج) زیرا شرایطی پیش نمی آید که چندین فرآیند قفلی را باز ببینند و وقتی قفل ورود به شرایط بحرانی باز است، فرآیندی از هسته های دیگر وجود ندارد که به طور موازی آن را باز ببیند و بخواهد وارد آن شود که شرط انحصار متقابل نقض شود. در نتیجه در تک هسته ای با اتمیک بودن دستورات مشکل حل میشود.

د) وقتی در چند هسته فرآیند ها دارند به صورت موازی اجرا می شوند ممکن است در لحظه ای یک قفل باز شود و چندین فرآیند آن را باز شده فرض کنند و بخواهند وارد آن شوند. در این شرایط خود سخت افزار با استفاده از ماژول هایی جلوی این کار را می گیرد و باعث می شود فقط یک دستور TAS, CAS به داده ها دسترسی پیدا کنند. در نتیجه انحصار متقابل حفظ می شود.