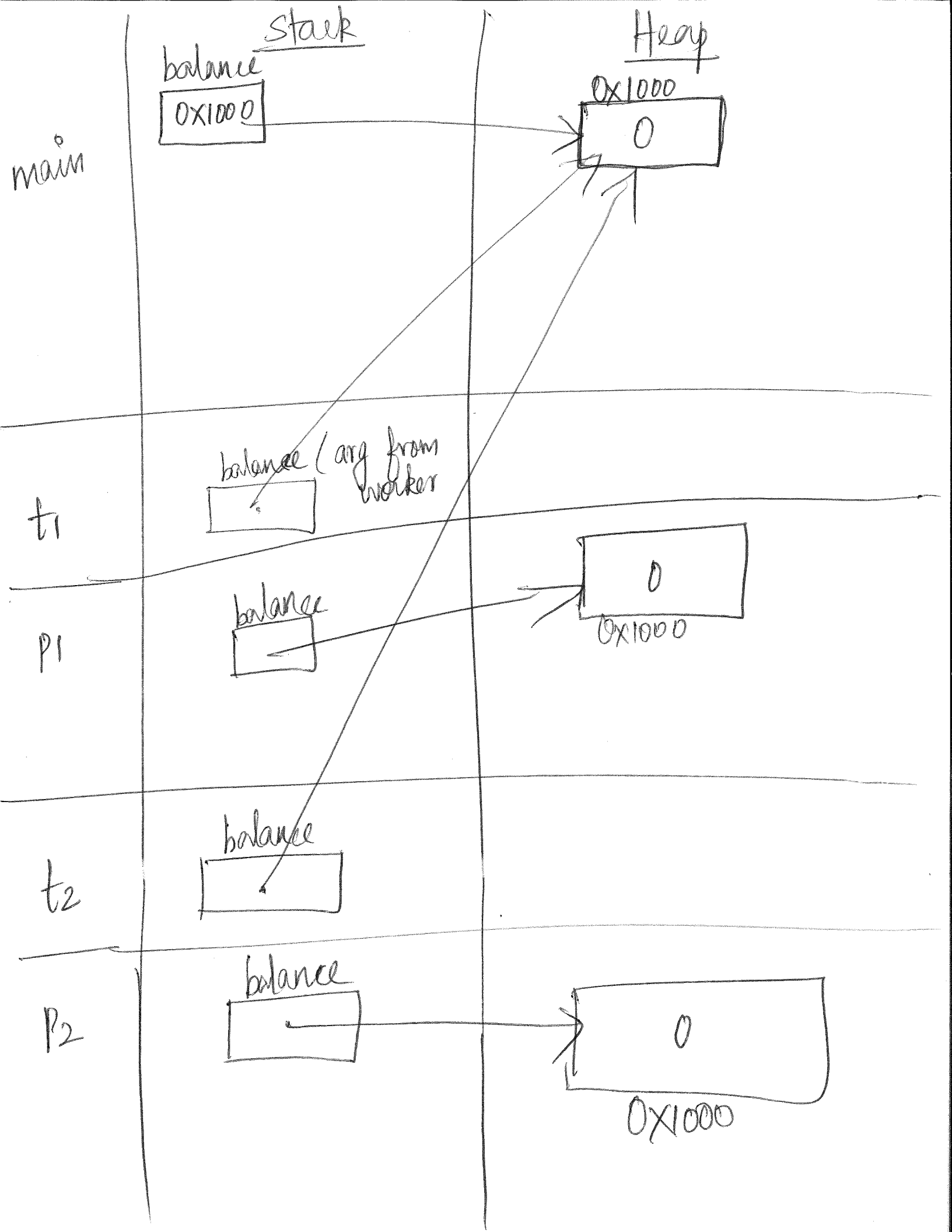
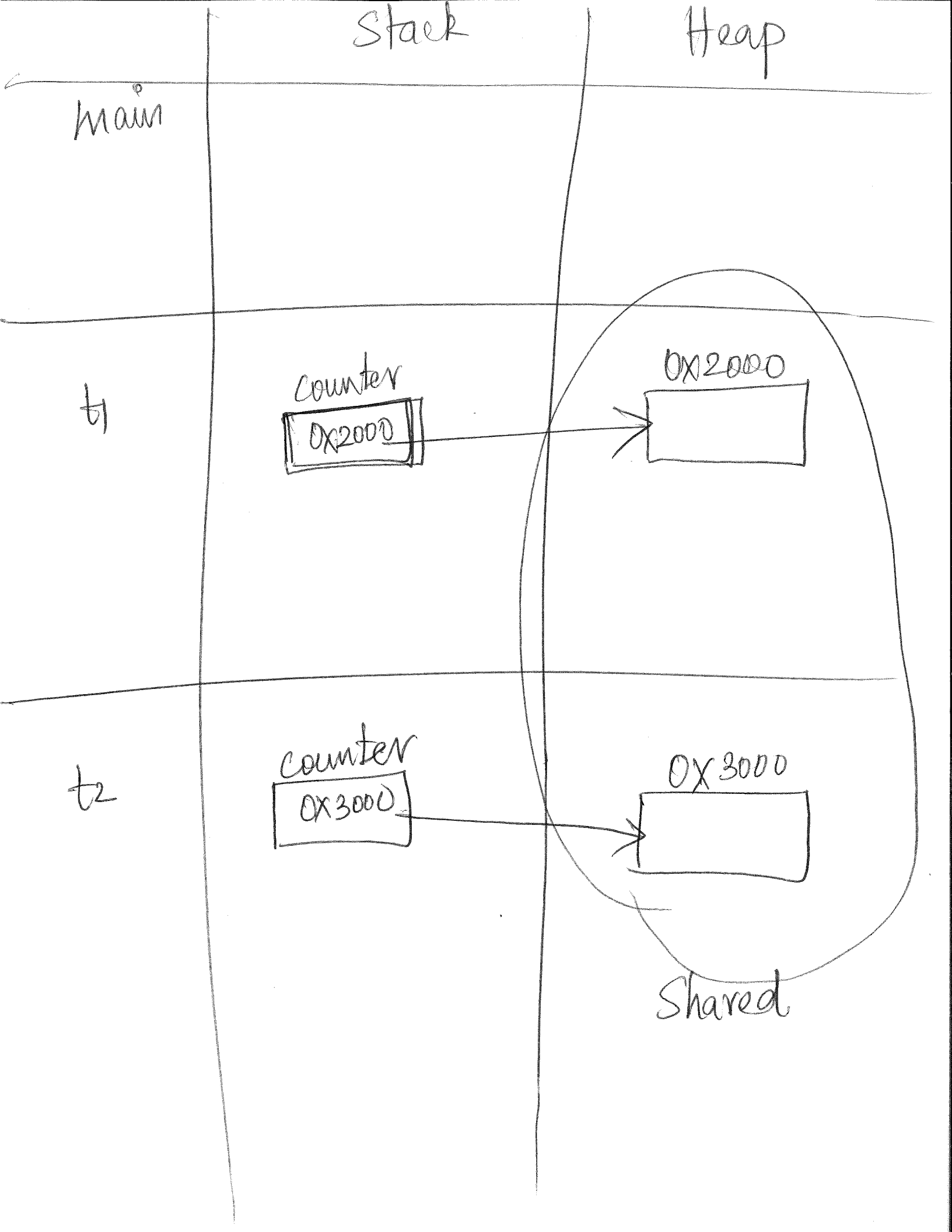
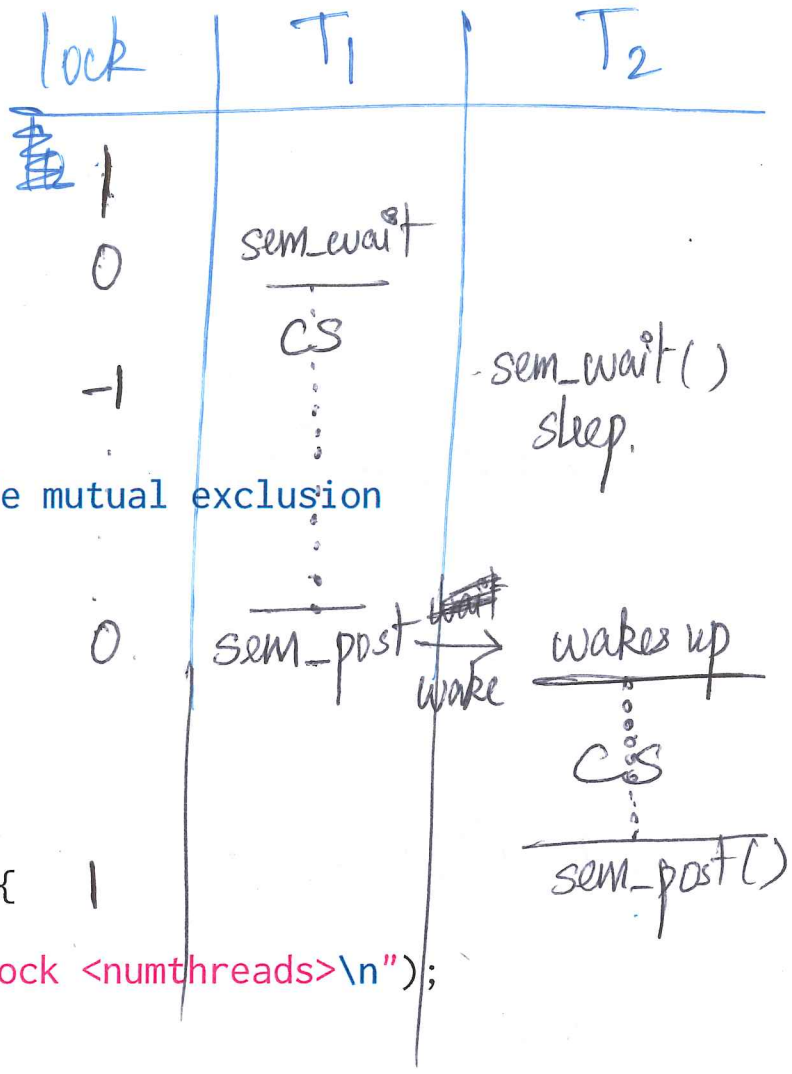
Midterm 2

Average = 78 % (96/120)

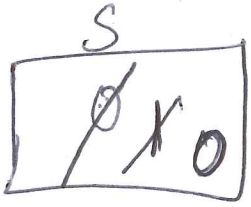

Stack
Heap
balance
0x1000
0x1000
0
main
balance (arg from worker
t1
0
balance
P1
0x1000
balance
t2
balance
P2
0
0x1000

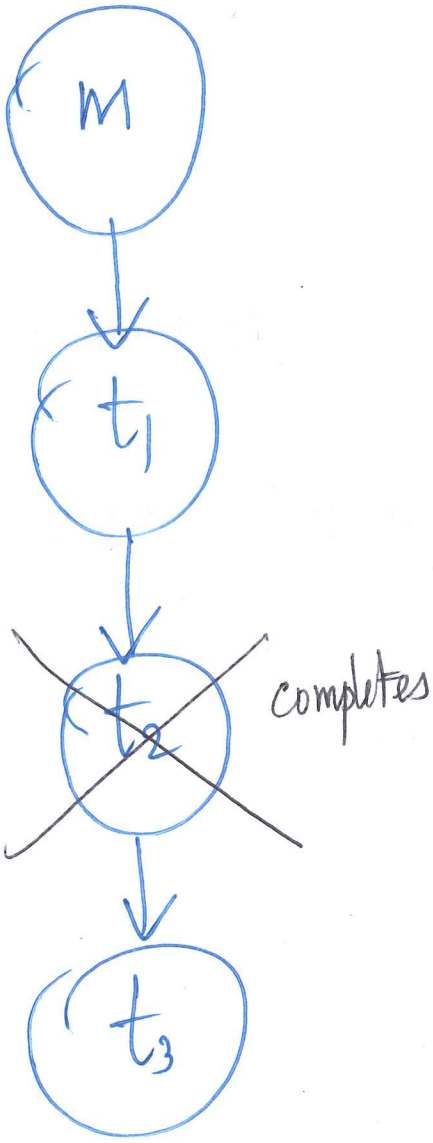

Stack
Heap
main
t1
Counter
0X2000
0X2000
t2
counter
0X3000
0X3000
Shared

```c
// Using a semaphore as a mutex lock!
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mythreads.h"
#define PMAX (100)

volatile static int counter = 0;
sem_t lock;

void *worker(void *arg) {
  int i;
  // add something here to provide mutual exclusion
  Sem_wait(&lock);
  for (i = 0; i < 1e6; i++)
    counter++;
  // something here: end mutex
  Sem_post(&lock);
  return NULL;
}

int main(int argc, char *argv[]) {
  if (argc != 2) {
    fprintf(stderr, "usage: sem-lock <numthreads>\n");
    exit(1);
  }
  int threads = atoi(argv[1]);
  if (threads > PMAX) {
    fprintf(stderr, "%d threads is the max\n", PMAX);
    exit(1);
  }

  pthread_t pid[PMAX];
  Sem_init(&lock, ???); // what value should we initialize here?
  int i;

  for (i = 0; i < threads; i++)
    Pthread_create(&pid[i], NULL, worker, NULL);

  for (i = 0; i < threads; i++)
    Pthread_join(pid[i], NULL);

  printf("counter: %d\n", counter);
  return 0;
}
```

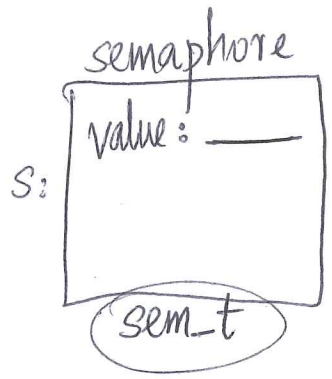


```c
// Using a semaphore for ordering!
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include "mythreads.h"

sem_t s;  // global, shared

void *child(void *arg) {
  printf("child\n");
  // something here
  sem_post(&s);
  return NULL;
}

int main(int argc, char *argv[]) {
  pthread_t p;
  printf("parent: begin\n");
  // init here
  sem_init(&s, 0);
  Pthread_create(&p, NULL, child, NULL);
  // something here
  sem_wait(&s);
  printf("parent: end\n");
  return 0;
}
```





```
1  //
2  // SEMAPHORE: PSEUDO-CODE
3  //
4  sem_init(sem_t *s, int initvalue) {
5      s->value = initvalue;
6  }
7
8  sem_wait(sem_t *s) {
9      s->value--;
10     while (s->value < 0)
11         put_self_to_sleep(); // put self to sleep
12 }
13
14 sem_post(sem_t *s) {
15     s->value++;
16     wake_one_waiting_thread(); // if there is one
17 }
18
19 //
20 // IMPORTANT: each is done atomically
21 // (i.e., body of post() and wait() happen all at once)
22 //
```



if

CS