Threads

- In a multithreaded program, all threads execute the same piece of code. They share the heap, but each thread has its own stack frame.

- main() function comprise a single, default thread. Threads other than the default one can be created by programmers.

- Threads in C

```c
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 5

typedef struct {
    int id;
} threadInfo;

void *hello_thread(void *threadinfo) {
    threadInfo *ti = (threadInfo *) threadinfo;
    printf("Thread %d saying Hello!\n", ti->id);
    pthread_exit(NULL);
}

int main () {
    int i;
    threadInfo ti[NUM_THREADS];
    pthread_t thread[NUM_THREADS];

    // Set up the parameters for each thread
    for (i = 0; i < NUM_THREADS; i++)
        ti[i].id = i;

    // Get the threads going
    for (i = 0; i < NUM_THREADS; i++)
        pthread_create(&thread[i], NULL, hello_thread, &ti[i]);

    // Join up with them. This will wait until they are done.
    for (i = 0; i < NUM_THREADS; i++)
        pthread_join(thread[i], NULL);

    return 0;
}
```
- Synchronization

  - To address the race condition
  - A thread must lock out other thread until it is done with its critical section
  - Critical section: the section of program where a thread read or write a shared variable

- Semaphore
  - Uses two atomic functions, P(s) and V(s)
  - P(s) - if s > 0 then assign s = s - 1; otherwise block the thread that called P.
  - V(s) - if a thread T is blocked on the semaphore s, wake up T; otherwise assign s = s + 1.

```c
/* initial state */
Semaphore empty = 1;
Semaphore full = 0;
Thing commonBuffer;

/* Thread A */
while (true) {
  Thing value = producer();
  P(empty);
  commonBuffer = value;
  V(full);
}

/* Thread B */
while (true) {
  P(full);
  Thing value = commonBuffer;
  V(empty);
  consumer(value);
}
```