Compare & Swap

lock = ~~0~~ 1.

→ takes value of lock
→ expected value
→ a new value.

```
int compare_and_swap (int *value ,int expected,
                                  int new_value){

    int temp = *value;
    if (*value == expected)
        *value = new_value;
    return temp;
}.
```

Mutex lock

↳ mutual exclusion lock.

available = true

```
do {
    acquire ~~section~~ lock
        critical section
    release lock
        remainder section
}while(true);
```

```
acquire () {
    while (!available)
        ; /* busy wait */
    available = false;
}

release () {
        available = true;
}.
```

```
* { pthread_mutex_t mutex; //
  { pthread_mutex_init(&mutex, NULL); //
```

→ equivalent to → available = true

Semaphore

→ by nature it is an integer

→ accepted only by wait() and signal().

→ when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

→

→ wait function এর কাজ semaphore এর value 1 কমানো।

→ signal function এর কাজ semaphore এর value 1 বাড়ানো।

```
wait(S){
  while (S<=0)
     ; //busy wait
     S--;
}
```

```
Signal (S) {
    S++;
}.
```

When, $\left\{ \begin{array}{l} S \rightarrow 1 \\ S \rightarrow 0 \end{array} \right.$ = works like mutex!

= T1, T2 cannot enter critical section.

binary semaphore

can use for synchonization.

```
funct1 () {
     printf("first");
     signal(s);
}
funct2 () {
T2 wait ();
      printf(" Second");
}.
```

⇒ Output: first
Second.

$S > 1 \rightarrow$ counting semaphore.

$S = 2$

T_3, T_2, T_1

```
sql = 'select * from student"
    wait(S)
  db.get (sql)
    signal(S).
```

⇒ At max 2 threads under the critical section.

Mid

Chapter 4 : Slide → 1 to 24.