Create thread using C

```
# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>
# include <pthread.h>

void *funcThread (void *arg);
int main (){
        pthread_t t1;
        pthread_create (&t1 ,NULL, funcThread , NULL);
        pthread_join (t1, NULL);

        return 0;

}
void *funcThread (void *arg){
        printf ('Enter thread: \n");
        for (int i=0; i<3; i++){
              printf ("thread : %d\n" ,i);
        }
        printf ("Done with thread...\n");

}.
```

→ variable memory address.

→ name of function

→ always null

→ parameters to the function.

Thread এর মধ্যে কোন parameter pass করতে চাইলে। fourth parameter এ memory address pass করবো।

Output

```
Entered thread:
    ⋮
Done with thread.
```

Elevated → 80 × 19 = 1520

BRAC → 50 × 4 = 200

Parking → 150

Parking → 50

Parking → 60

Total : 1980

$$\begin{array}{r} 1980 \\ +\ 50 \\ +\ 80 \\ \hline 2110 \end{array}$$

cse 420 ✓

HUM 101 ✓

ANT 101

mat 215

??? CSE 425

Thread এ pointer input নেই + pointer output দেই।

```
int * func_thread (int *v);
int *t_ret;
int main () {
        pthread_t t1;
        int n=5;
        pthread_create (&t1, NULL, func_thread, &n);
        pthread_join (t1, &t_ret);
        printf ("Thread returned : %d\n",*t_ret);
        return 0;
}
int * func_thread (int *v) {
        *v = *v * 5;
        return v;
}
```

(i) memory address

(ii)

(iii)

(iv)

as int receive হিসেবে চাচ্ছি তাই int pointer pass করেছি

pointer এর Bhitorer value 5 diye multiply হয়েছে। 5×5 = 25.

Output :

Thread returned 25.

*v = pointer → address.

v = value.

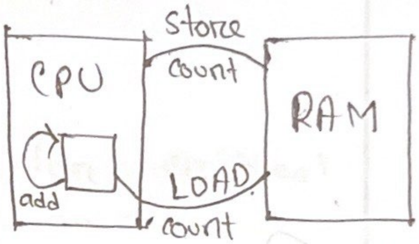
T1 T2 Context switch can happen at any time.

Count = count + 1

i) Load

ii) Add

iii) Store



count = 0
count = 1
count = 2.

✓ ✗

(ii) T1 T2 (i)

count : 1 count : 2

count : 2

i এর value বাড়ালি এমন weired behaviour করে।

i এর value কম থাকলে এমন হওযার chance কম।

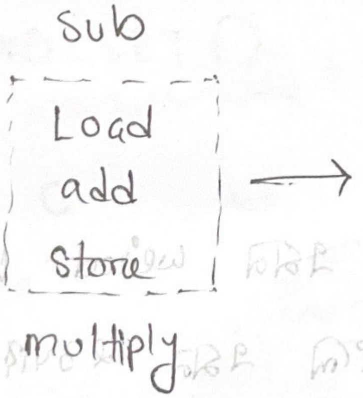
if we can control the context switch then, we solve the problem.

Subject ____________ Sat ○ Sun ○ Mon ○ Tue ○ Wed ○ Thu ○ Fri ○ Date : / /

Process Synchronization

When multiple process manipulates a variable, a condition prevails, called Race condition.

সেই section এর মাঝে context switch না করতে দিবো, সেই section of code কে বলে critical section.



→ critical section.
allowed only one thread at a time.
↳ called mutually exclusion.

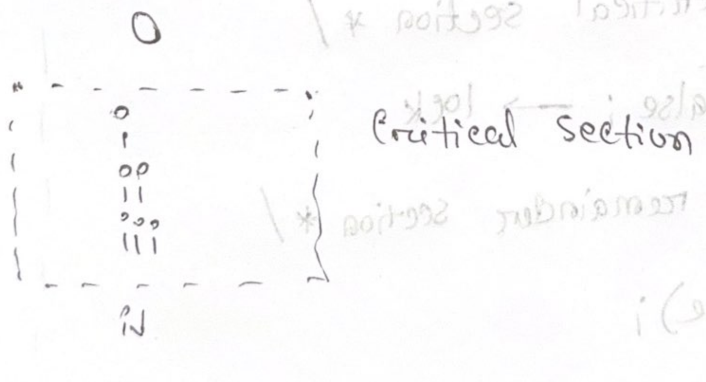
When a critical section is empty and a thread wants to enterr the critical section, we should let it enterr. This is called Progress.

Hardware based solution for critical section problem :

counterr ++ counterr --

দুটো পুরোটা হবে creitical section.

Test_AND_SET ()



```
boolean test_and_set (boolean *target) {
        boolean rv = *target;
        *target = true;

        return rv;
```

atomic in nature. means, we cannot break the function. And, ~~afterr~~ in this function, context switch cannot happen. Either it will run or, it won't.

→ returns the previous value.

→ make it true forcefully.

```
do {
   while (test_and_set (&lock)) → entry.
     ; /* do nothing */

     /* critical section */
   lock = false; → lock

     /* remainder section */
} while (true);
```

Handware, ensures that, context switch won't happen here in the atomic function.

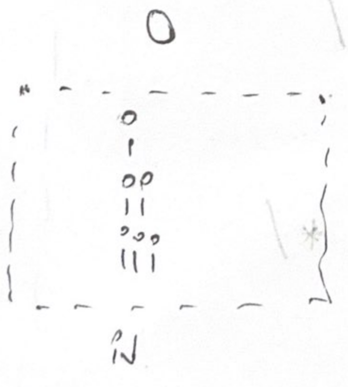
When a critical section is empty and a threed wants to entere the critical section, we should let it enter. This is called Progress.

Handware based solution for critical Section problem:

counter ++ counter --

দুই দুইটা রাখ critical section.

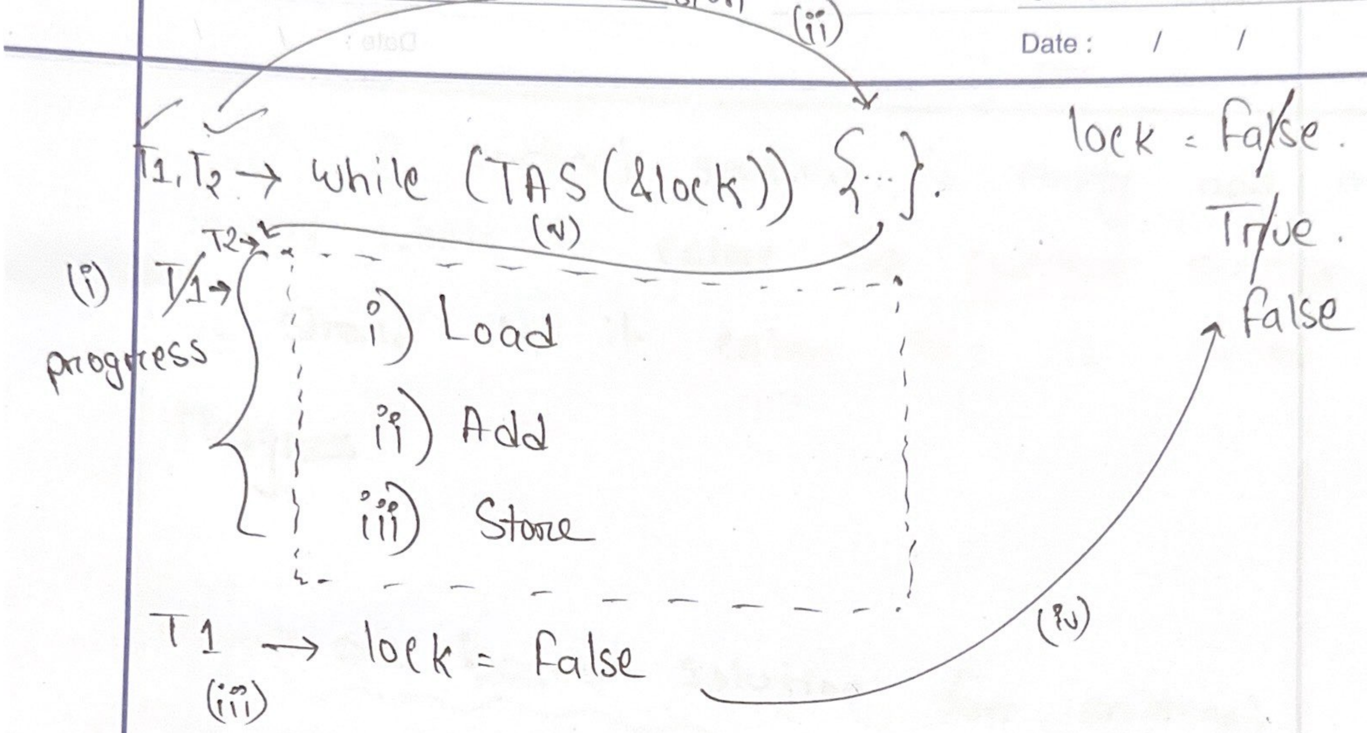
Test_AND_SET()



Subject mutual exclusion

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /



iterating in a empty while loop is called spinning.

while (true) → infinite while loop.