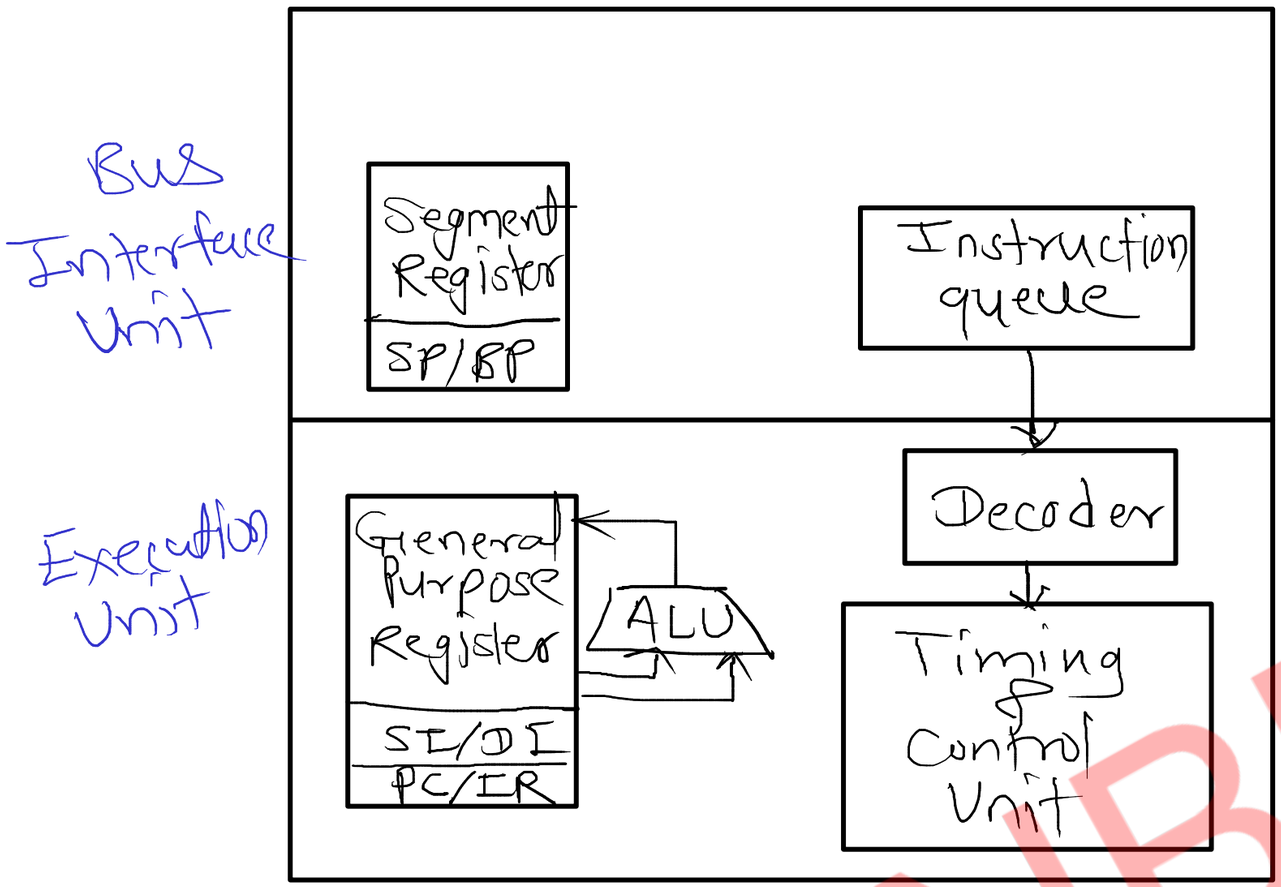
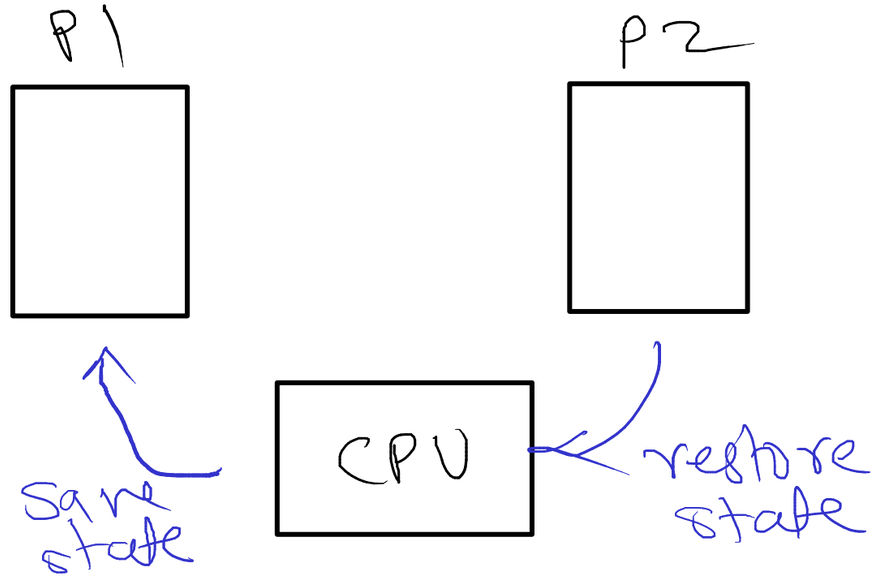


Execution Context :

— values of CPU registers

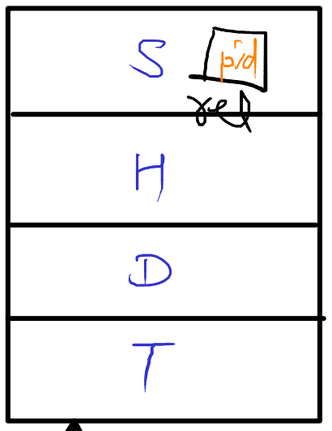
Context switching :

— save execution context of one process & restore execution context of another process.



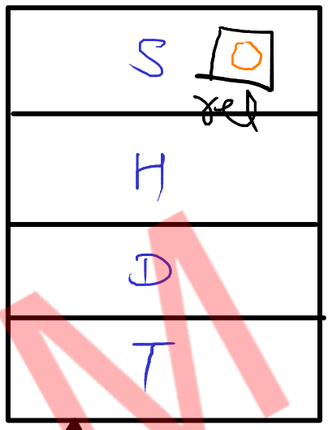
Parent process

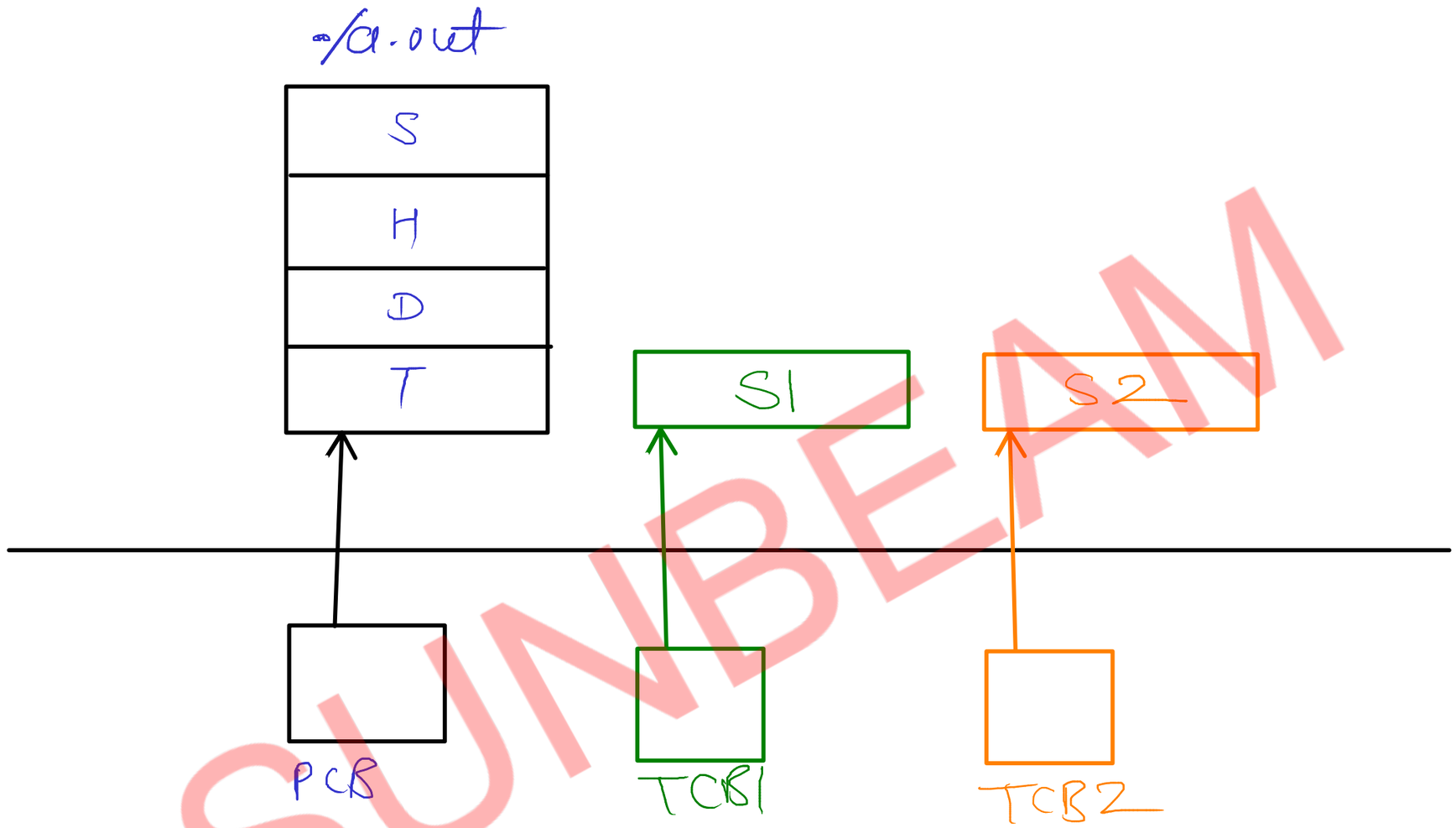
./a.out



fork()

child process





Semaphore

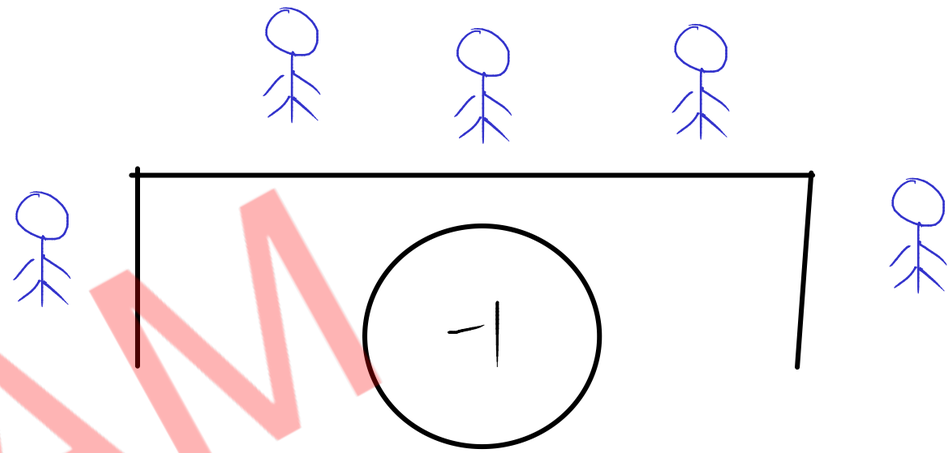
- semaphore is internally a counter

Dec:
1) decrement counter
2) if counter < 0, then block current thread/process.

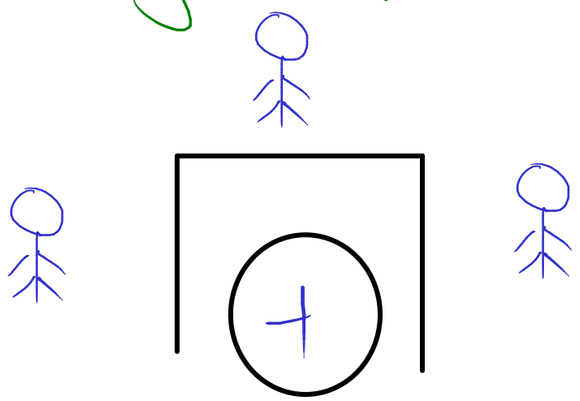
Inc:
1) increment counter
2) if any process/thread is blocked on this semaphore, wake up one.

create — sem_init()
dec — sem_wait()
inc — sem_post()
destroy — sem_destroy()

Counting semaphore



Binary semphore



sem_init(&s, 0, 1)

void inc(void)
{
for( )
{
sem_wait(&s)
count++;
sem_post(&s)
}
}

void dec(void)
{
for( )
{
sem_wait(&s)
count--;
sem_post(&s)
}
}

sem_destroy(&s)

Mutex

- Mutual Exclusion — one at a time
- Mutex is similar to binary semaphore.
- Mutex operations - lock(), unlock()
- The process who will lock the mutex, will become owner of that mutex.
- Only owner can unlock the mutex.

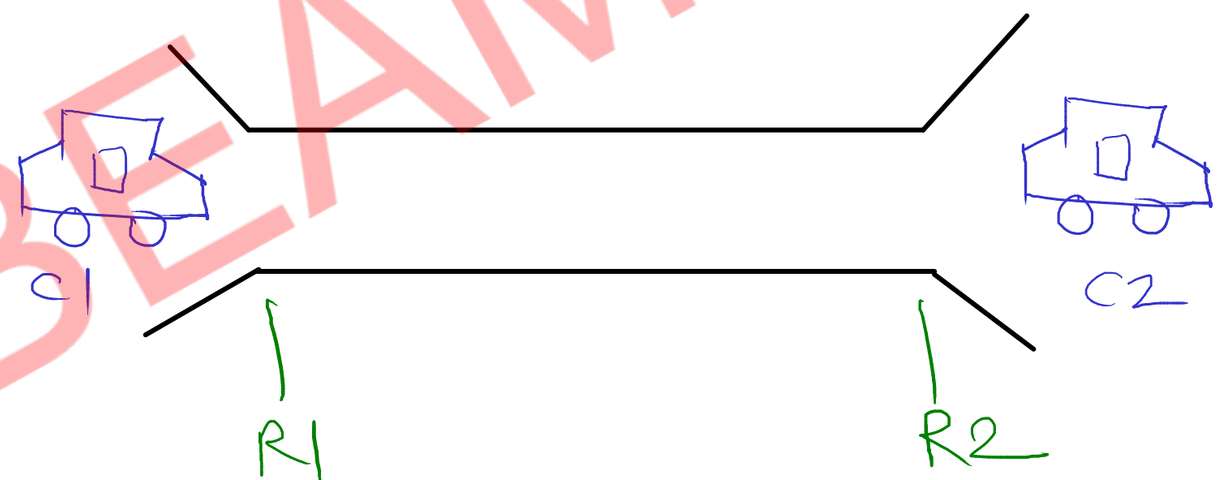
pthread_mutex_create() — create
pthread_mutex_lock() — lock
pthread_mutex_unlock() — unlock
pthread_mutex_destroy() — destroy.

DeadLock

— indefinit waiting of process for some common resource.

— there are four conditions of deadlock. & when all four conditions will be true, at same time deadlock will occur.

1) Mutual exclusion
2) No preemption
3) Hold & wait
4) Circular wait.



Prevention:

— while implementing code of OS, it is ensured that, any one condition out of 4 is always false

Avoidance:

1) resource allocation graph
2) Banker's algorithm
3) Safe state algorithm

Recovery:

1) resource preemption
2) Process termination