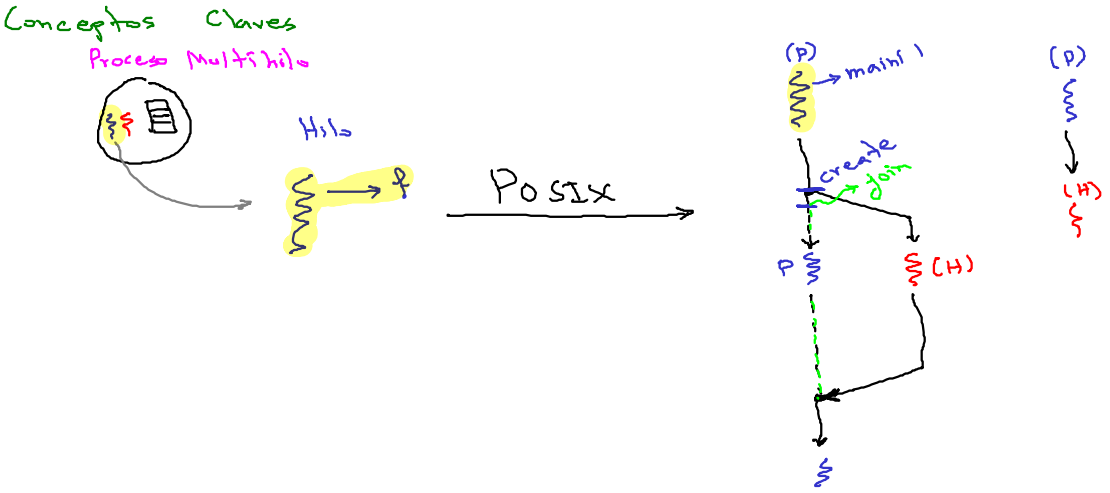
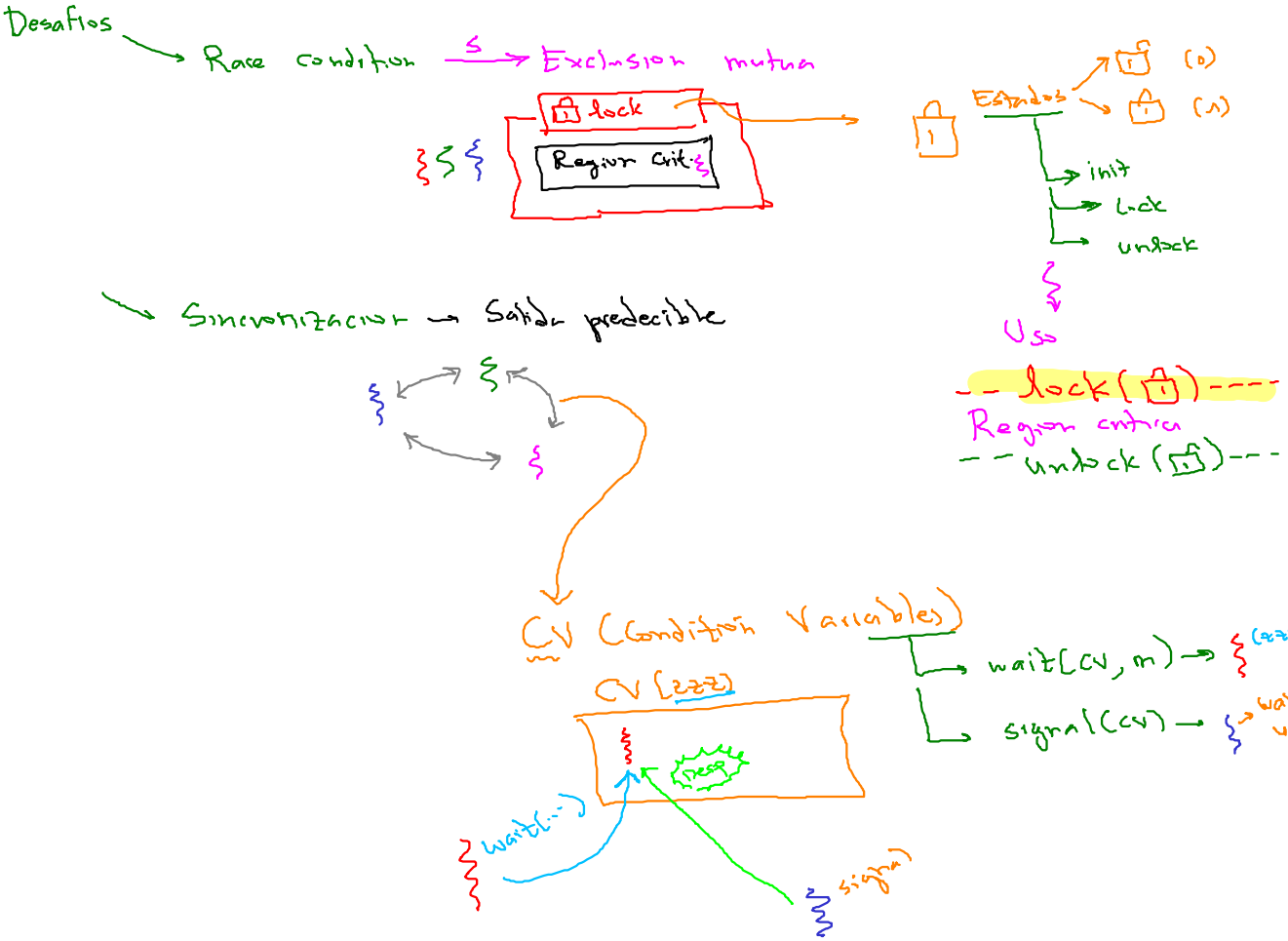
17/06/2025 - Sistemas Operativos - Ude@

1. Repaso





2. Sincronización

Implementación del join - Caso 3: Los problemas de las 2 implementaciones anteriores ya es solucionado

```c
int done = 0;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;
void *child(void *arg) {
  printf("child\n");
  thr_exit();
  return NULL;
}
int main(int argc, char *argv[]) {
  printf("parent: begin\n");
  pthread_t p;
  pthread_create(&p, NULL, child, NULL);
  thr_join();
  printf("parent: end\n");
  return 0;
}
```

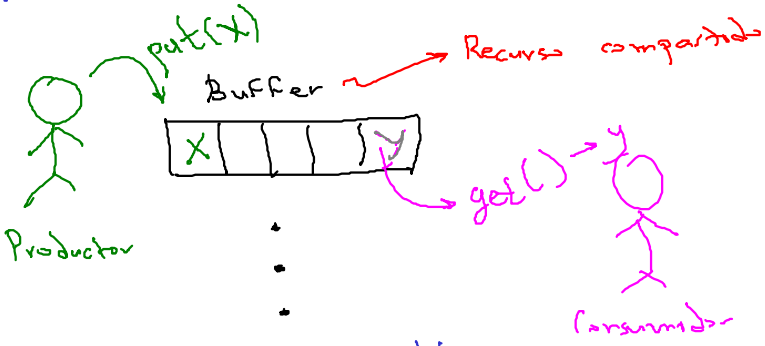
```c
void thr_exit() {
  pthread_mutex_lock(&m);
  done = 1;
  pthread_cond_signal(&c);
  pthread_mutex_unlock(&m);
}
```

L
RC
UL

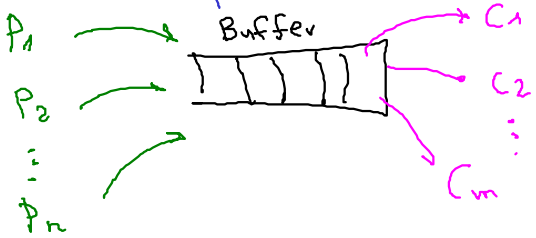
```c
void thr_join() {
  pthread_mutex_lock(&m);
  while (done == 0)
    pthread_cond_wait(&c, &m);
  pthread_mutex_unlock(&m);
}
```

L
UL

3. Problema del Productor consumidor.

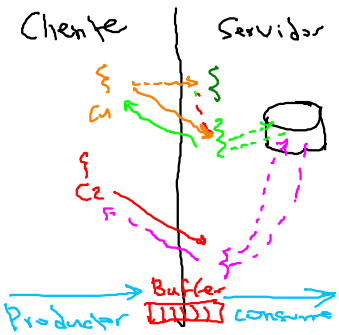


Generalización del problema



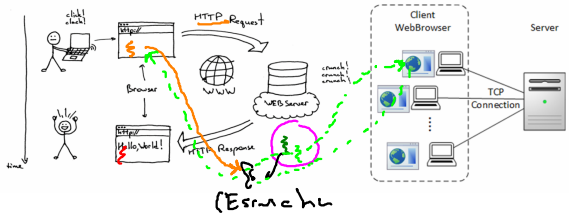
4. Casos de uso:

1. Supermercado ✓
2. Negocio de comidas Rapidas ✓
3. Servidor Multihilos ✓



Servidor multi-hilo

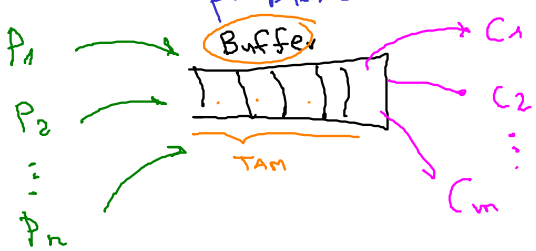
- Un **productor** ingresa **`http request`** en una **cola** de trabajo.
- El **consumidor** retira requerimientos de la **cola** y los procesa.



4. Consola de linea de comandos ✓

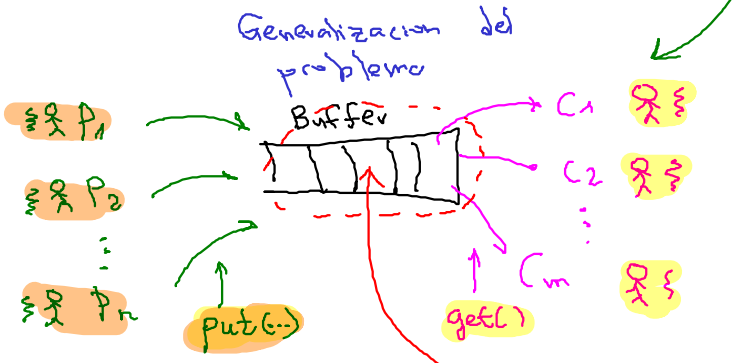
```
grep foo file.txt | wc l
```

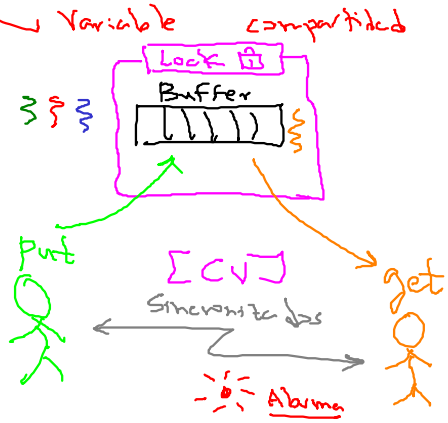
Generalización del problema



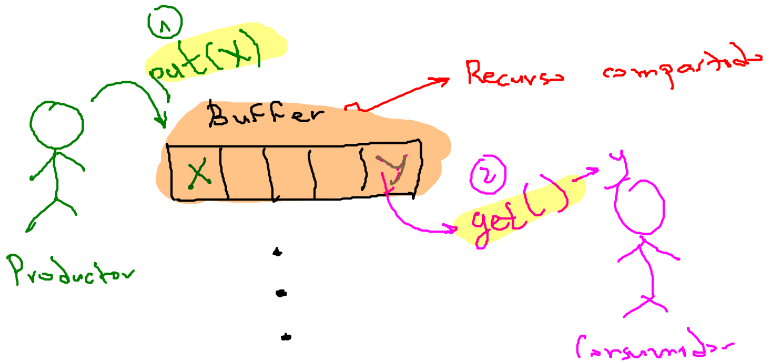
PROBLEMA DE PRODUCTOR - CONSUMIDOR CON BUFFER LIMITADO

5. Como llevar esto a codigo [Tools: ① Hilo → f(...) ② Lock (Excl. Mutua.) ③ CV (Condition Variable)]





6. Implementación:

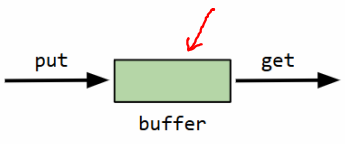


* Secuencial

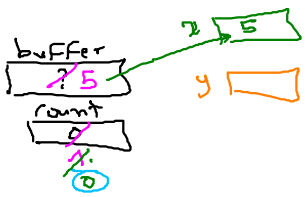
```
int buffer;
int count = 0; // initially, empty

① void put(int value) {
  assert(count == 0);   vacio?
  count = 1;
  buffer = value;
}

② int get() {
  assert(count == 1);   lleno?
  count = 0;
  return buffer;
}
```

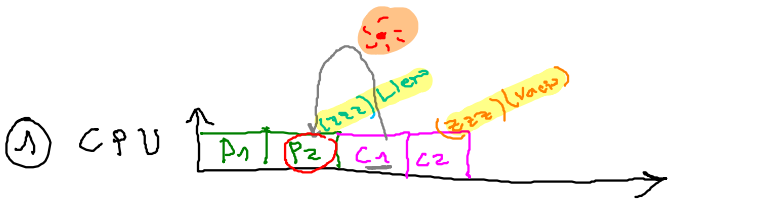


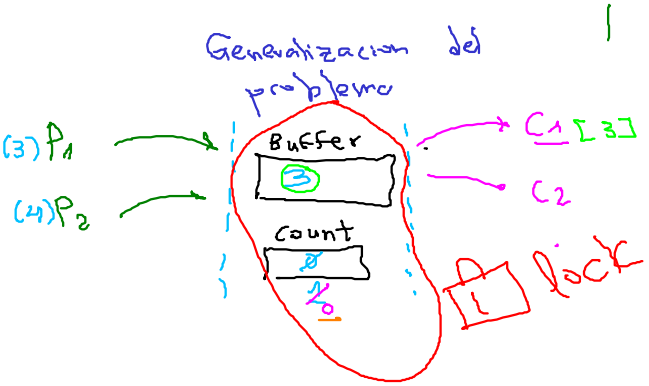
count → flag (ocupacion)
count = 0 [vacio]
count = 1 [lleno]



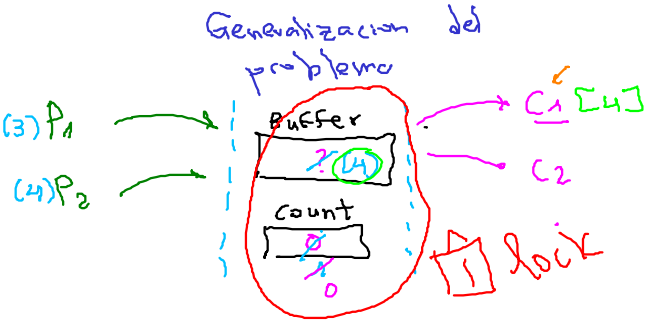
* Realidad → Varias tareas a la vez

Multihilo ({ { {) → lock
→ CV (wait / signal)



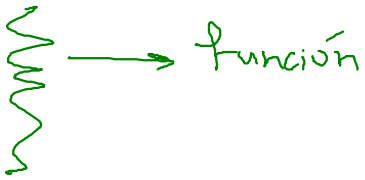


zzz → (wait)
(signal)



Al usar hilos

① Hilos → Tarea

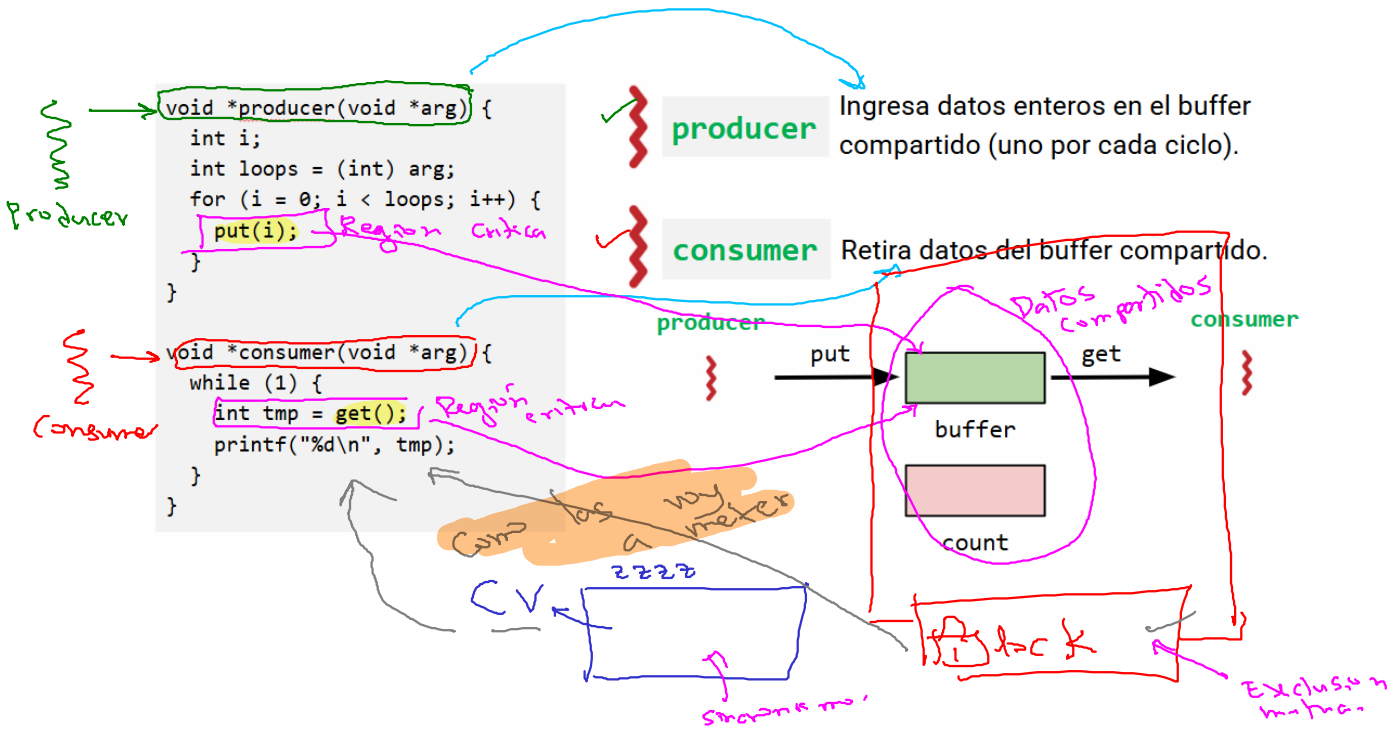



```c
void *producer(void *arg) {
  int i;
  int loops = (int) arg;
  for (i = 0; i < loops; i++) {
    put(i);
  }
}

void *consumer(void *arg) {
  while (1) {
    int tmp = get();
    printf("%d\n", tmp);
  }
}
```

producer Ingresa datos enteros en el buffer compartido (uno por cada ciclo).

consumer Retira datos del buffer compartido.



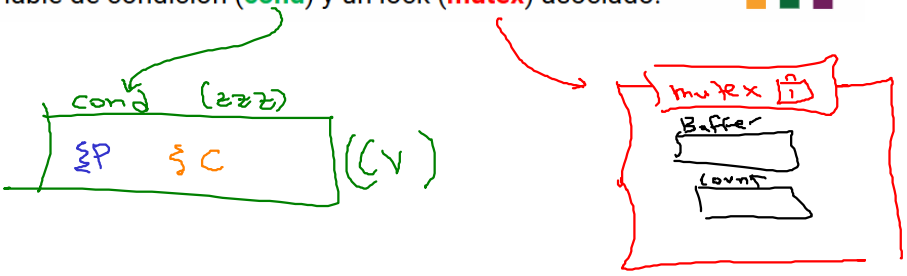
- **Productor/consumidor: Unica CV y sentencia if**
- Productor/consumidor: Unica CV y sentencia while
- Productor/consumidor – Single buffer: Usar 2 variables de condición y un while

Use **variables de condición** (CV) para hacer que los **consumidores esperen** cuando no haya nada que consumir (buffer vacío) y los **productores esperen** cuando el buffer se encuentre lleno.

```c
int loops; // must initialize somewhere...
cond_t cond;
mutex_t mutex;
void *producer(void *arg) {
  int i;
  for (i = 0; i < loops; i++) {
    pthread_mutex_lock(&mutex);            // p1
    if (count == 1)                        // p2
      pthread_cond_wait(&cond, &mutex);    // p3
    put(i);                                // p4
    pthread_cond_signal(&cond);            // p5
    pthread_mutex_unlock(&mutex);          // p6
  }
}
```

```c
void *consumer(void *arg) {
  int i;
  while (1) {
    pthread_mutex_lock(&mutex);            // c1
    if (count == 0)                        // c2
      pthread_cond_wait(&cond, &mutex);    // c3
    tmp = get();                           // c4
    pthread_cond_signal(&cond);            // c5
    pthread_mutex_unlock(&mutex);          // c6
    printf("%d\n", tmp);
  }
}
```

- Se usa una única variable de condición (**cond**) y un lock (**mutex**) asociado.



Sera que sirve? → Pruebas de Escritorio

Caso 1 ✓

Caso 2 ✓

⋮

Caso n ✗ → Otra solución