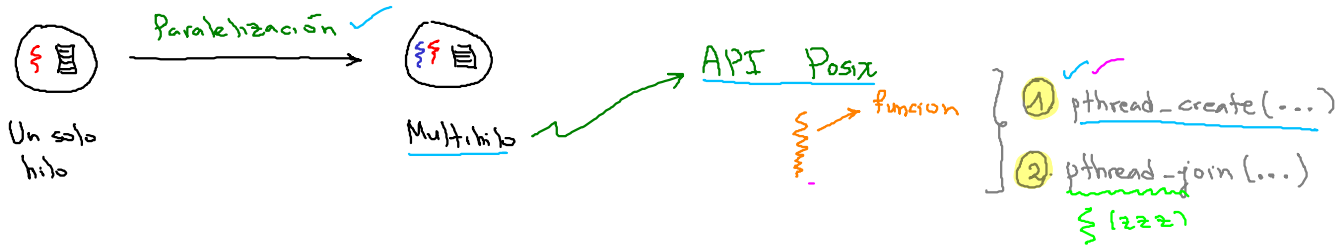
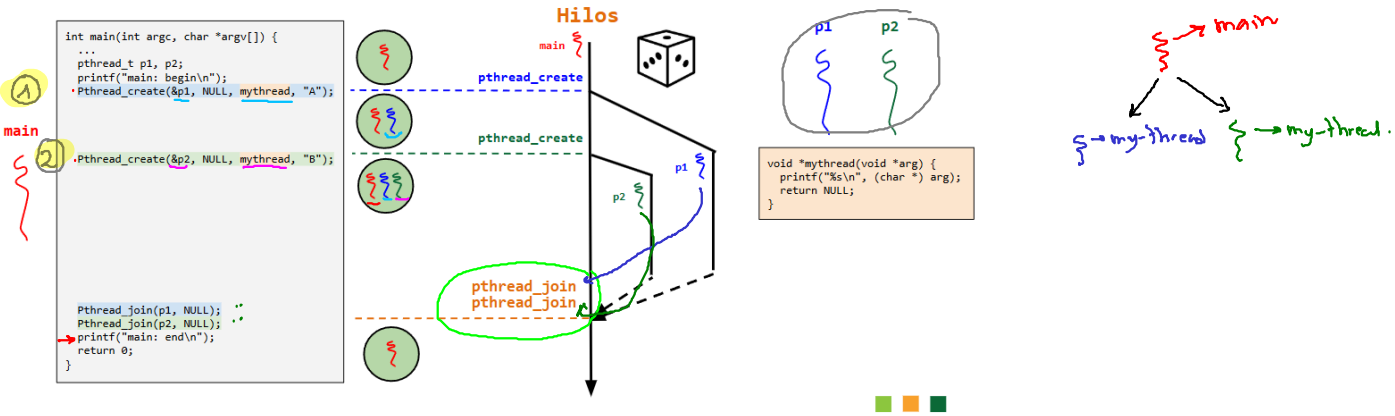
12/06/2025 - Sistemas Operativos - Ude@

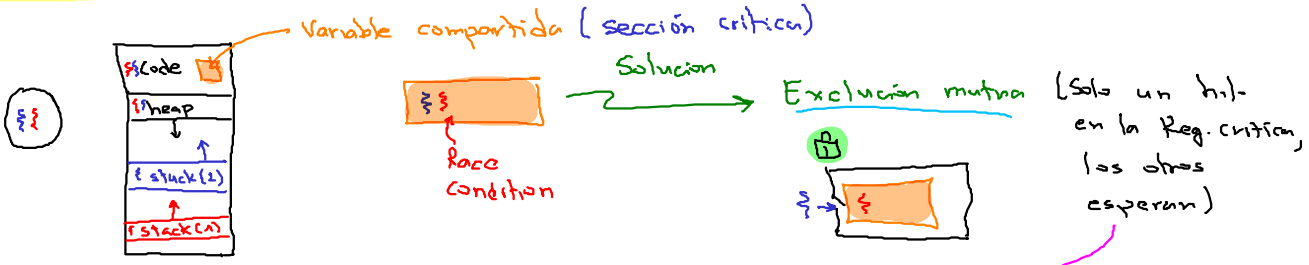
1. Repaso.



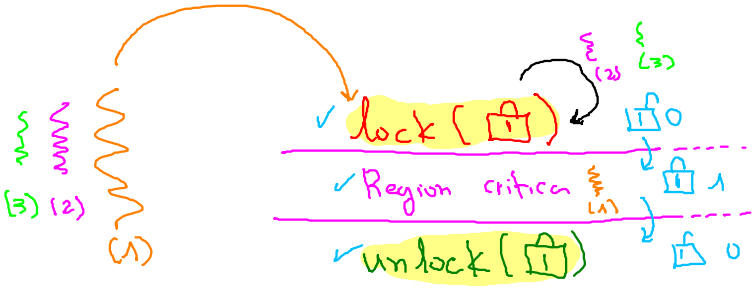


Cuando se paraleliza nos enfrentamos a nuevos desafíos (problemas)

1 Race condition

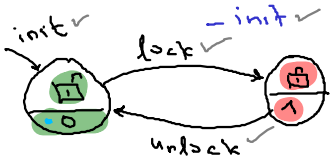


Se implementa usando locks (candados)



Estados:
- Free: 0
- held: 1

Funciones
- adquire = Lock
- release = unlock
- init



2 Orden Aleatorio (Problemas de sincronización)

→ pthread_join

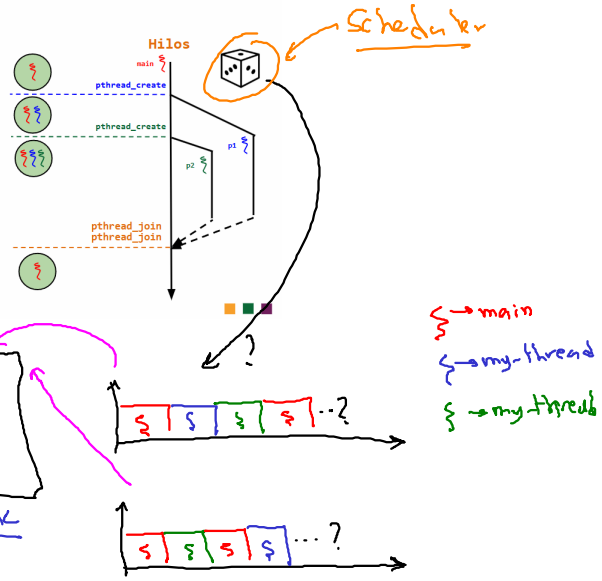
Ejemplo - Orden de ejecución

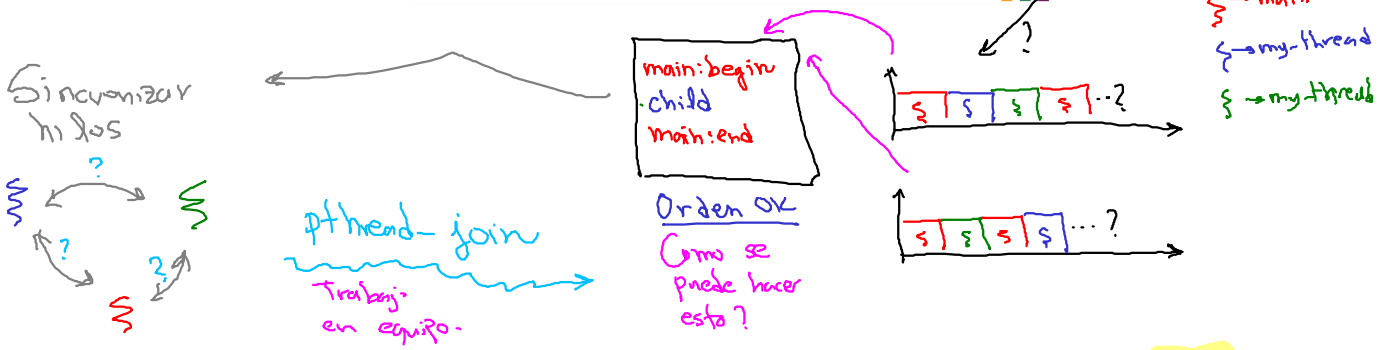
- **Pregunta**: ¿Cual hilo se ejecuta primero?
- **Respuesta**:
  - El orden es indeterminado (aleatorio).
  - Depende del scheduler.

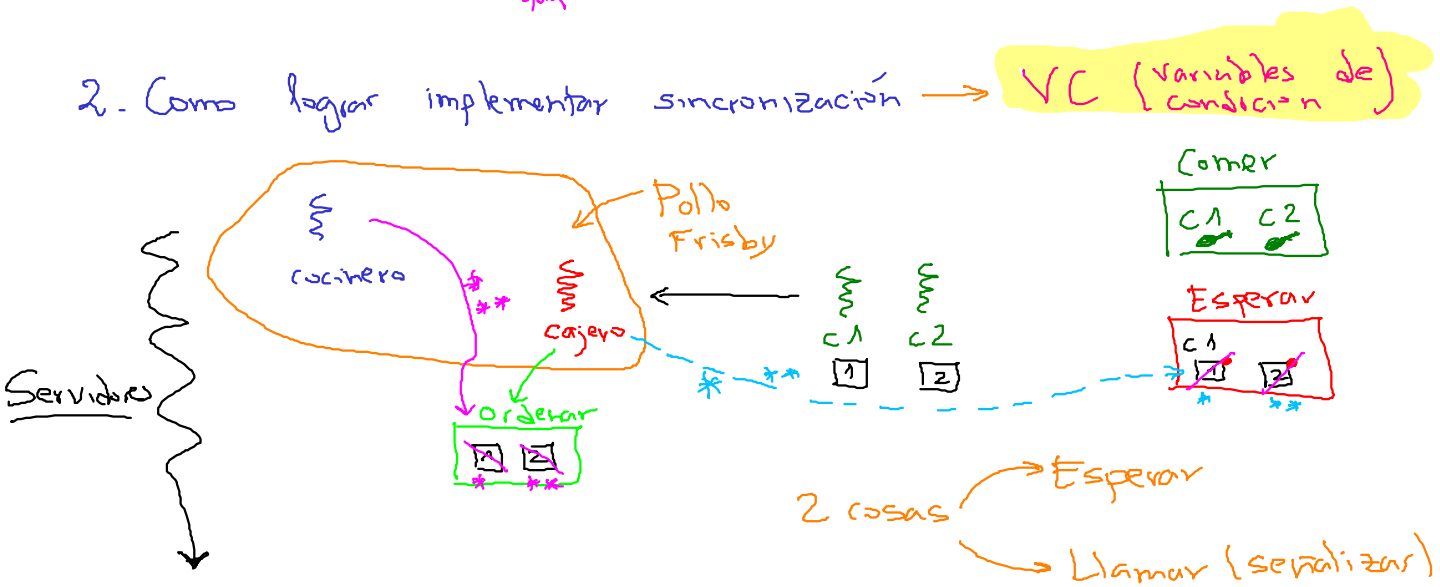
La salida es no determinista (Aleatoria)

Ejemplo - Orden de ejecución

```
13  int main(int argc, char *argv[]) {
14      if (argc != 1) {
15      fprintf(stderr, "usage: main\n");
16      exit(1);
17      }
18
19      pthread_t p1, p2;
20      printf("main: begin\n");
21      Pthread_create(&p1, NULL, mythread, "A");
22      Pthread_create(&p2, NULL, mythread, "B");
23      // join waits for the threads to finish
24      Pthread_join(p1, NULL);
25      Pthread_join(p2, NULL);
26      printf("main: end\n");
27      return 0;
28  }
```

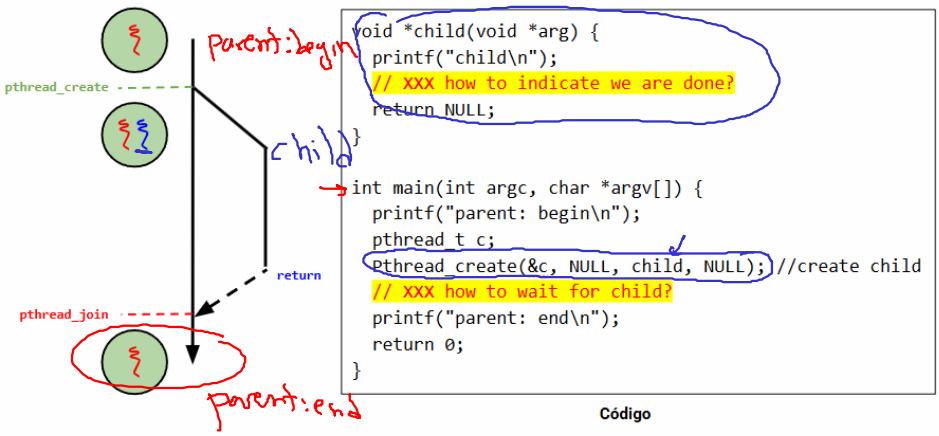






¿Cómo implementar el orden de ejecución deseado?

- **¿Cómo implementar el join?** (Para lograr el orden de ejecución deseado).




```
void *child(void *arg) {
  printf("child\n");
  // XXX how to indicate we are done?
  return NULL;
}

int main(int argc, char *argv[]) {
  printf("parent: begin\n");
  pthread_t c;
  Pthread_create(&c, NULL, child, NULL); //create child
  // XXX how to wait for child?
  printf("parent: end\n");
  return 0;
}
```

Código

main (Padre)

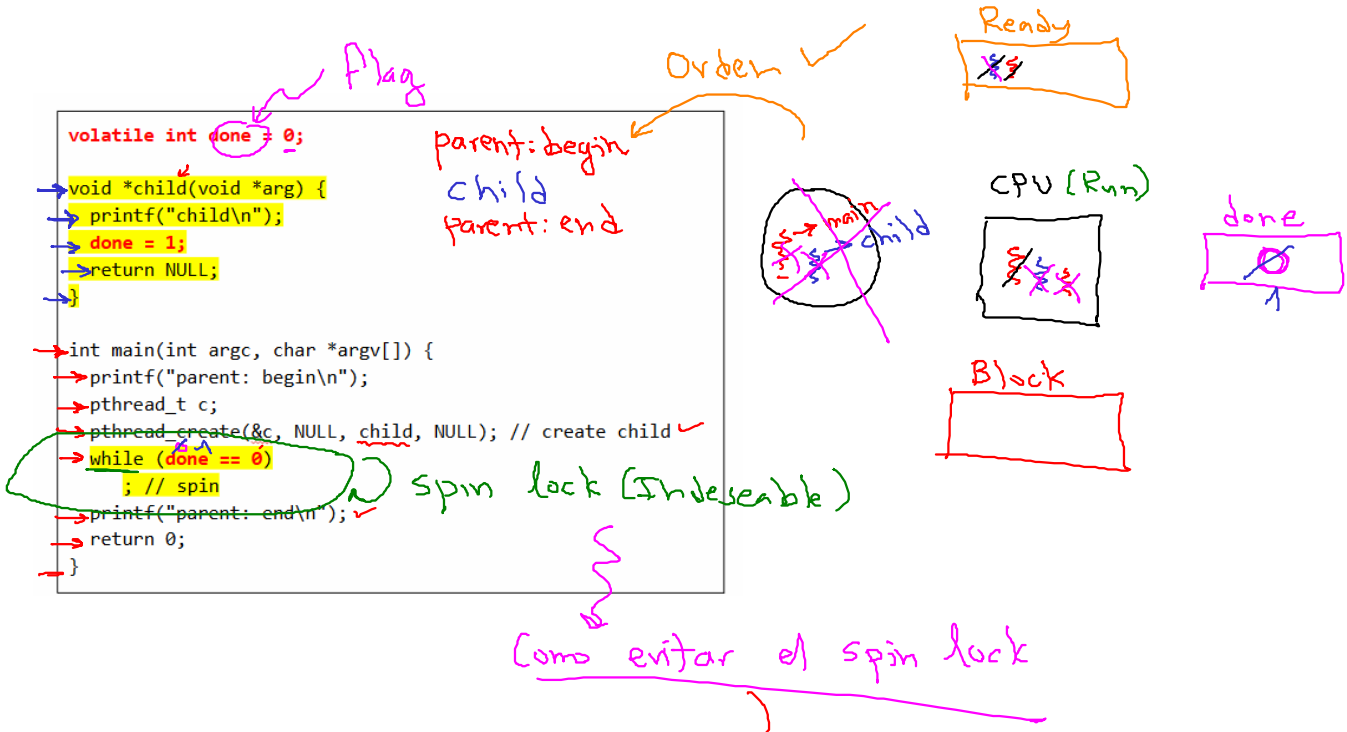
child (Hijo)

```
parent: begin
child
parent: end
```

Salida deseada

Como lograr que esta siempre sea la salida?

```c
volatile int done = 0;

void *child(void *arg) {
    printf("child\n");
    done = 1;
    return NULL;
}

int main(int argc, char *argv[]) {
    printf("parent: begin\n");
    pthread_t c;
    pthread_create(&c, NULL, child, NULL); // create child
    while (done == 0)
        ; // spin
    printf("parent: end\n");
    return 0;
}
```



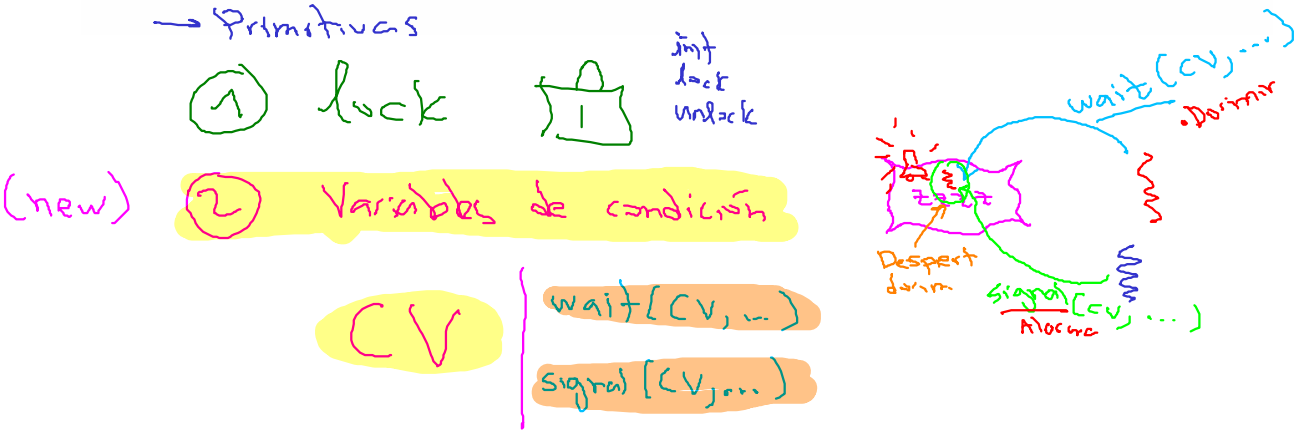
Variables de condición (Condition Variables - CV)

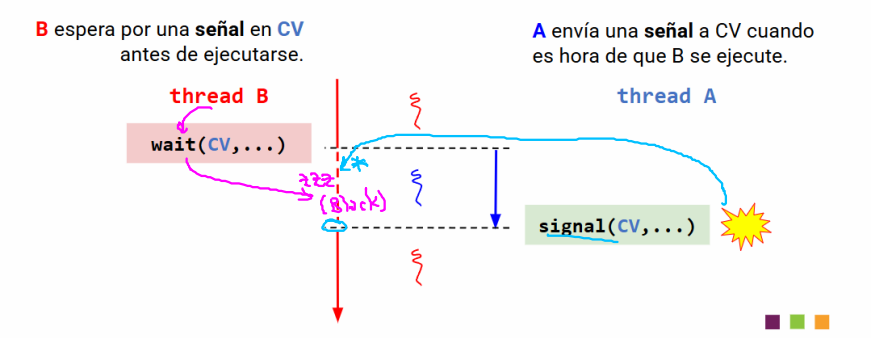
Variables de condicion (CV)

Idea clave: ¿Cómo esperar por una condición?

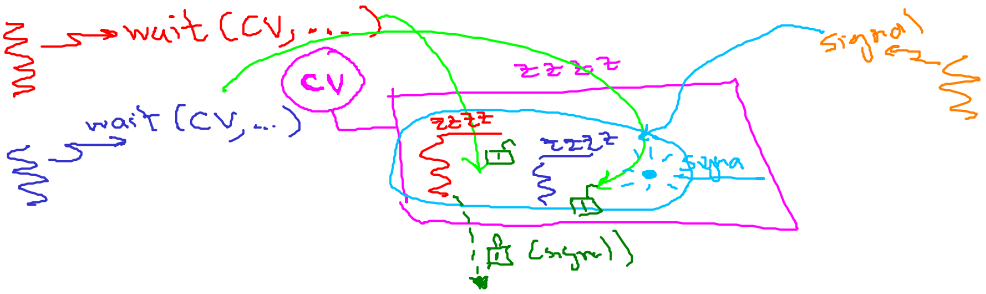
- **Espera de una condición:** En programas multihilo, es a menudo útil que un hilo espere hasta que alguna condición se vuelva verdadera antes de proseguir.
- **Solución ineficiente – Spinning:** Una solución es esperar validando continuamente hasta que la condición se vuelva verdadera (**spinning**) sin embargo esto es muy ineficiente debido al gasto de CPU.

¿Cómo implementar un hilo que espere por una condición?





3. Variables de Condicion y Funciones wait (zzz) y signal (despertar)



Operaciones de las variables de condición

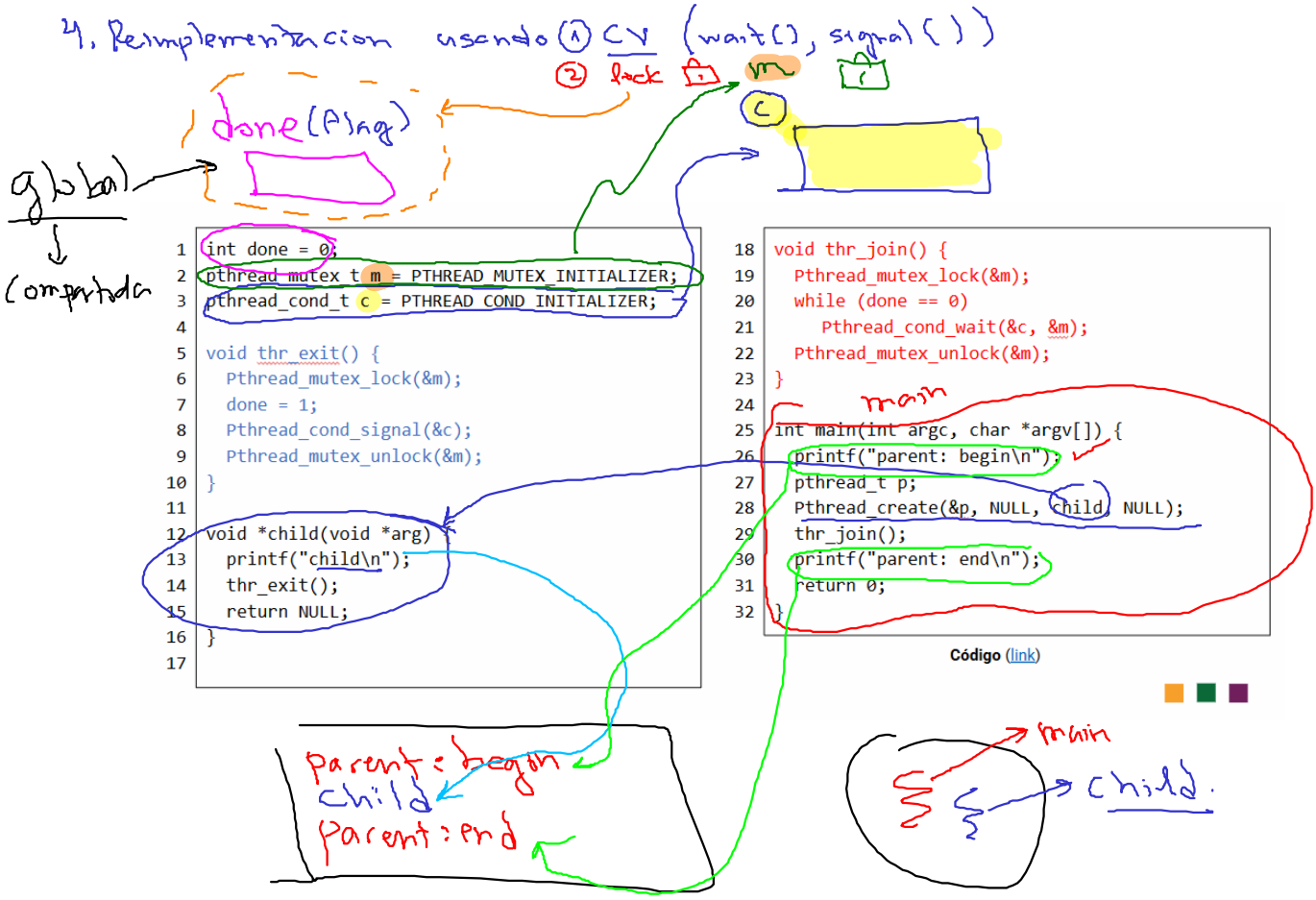
wait(cond_t *cv, mutex_t *mutex)

- Un hilo se pone en una cola (a dormir) a la espera de que un estado deseado suceda.
- Recibe un mutex como parámetro.
- Cuando se llama el wait el lock es liberado y se pone el hilo a dormir.

signal(cond_t *cv)

- Despierta un único hilo de los que se encuentran en la cola de espera (**if >= 1** el hilo está esperando).
- Si no hay hilos esperando, solo retorna sin hacer nada.
- Cuando el hilo se despierte se debe adquirir de nuevo el lock.

4. Reimplementacion usando (1) CV (wait(), signal()) (2) lock



Código (link)

Orden 1

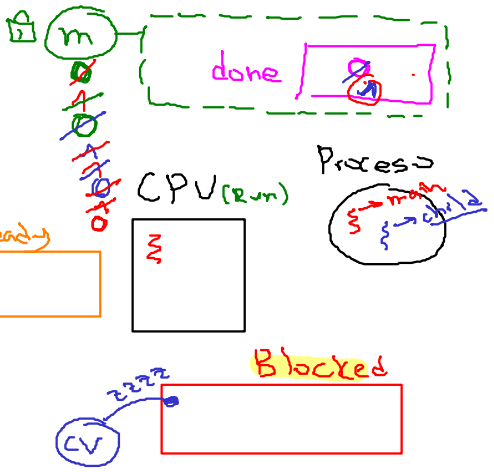
```c
int done = 0;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;

void thr_exit() {
    Pthread_mutex_lock(&m);
    done = 1;
    Pthread_cond_signal(&c);
    Pthread_mutex_unlock(&m);
}

void *child(void *arg) {
    printf("child\n");
    thr_exit();
    return NULL;
}
```

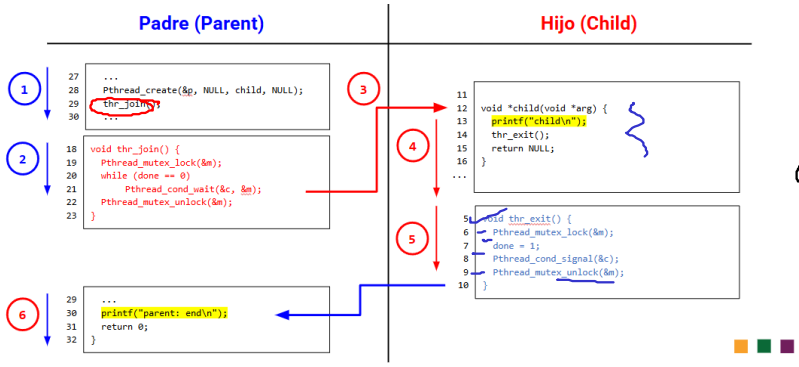
```c
void thr_join() {
    Pthread_mutex_lock(&m);
    while (done == 0)
        Pthread_cond_wait(&c, &m);
    Pthread_mutex_unlock(&m);
}

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;
}
```

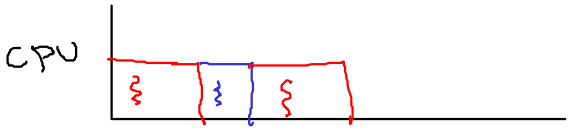
Output:

```
parent: begin
child
parent: end
```

Código (link)







Orden 2

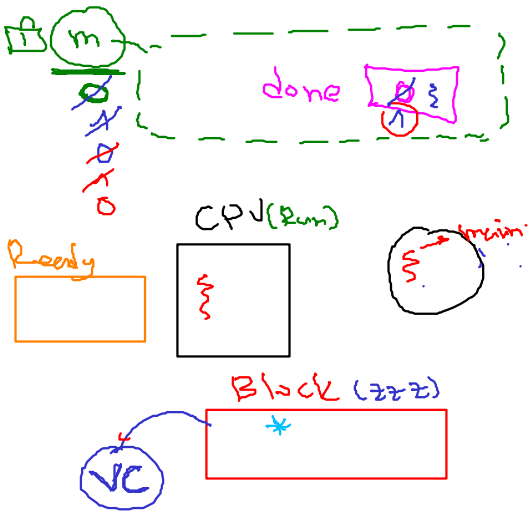
```c
int done = 0;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;

void thr_exit() {
    Pthread_mutex_lock(&m);
    done = 1;
    Pthread_cond_signal(&c);
    Pthread_mutex_unlock(&m);
}

void *child(void *arg) {
    printf("child\n");
    thr_exit();
    return NULL;
}
```

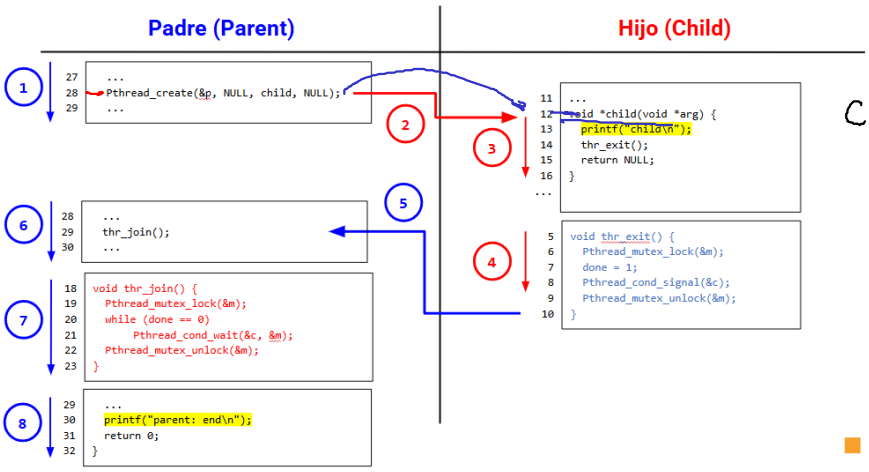
```c
void thr_join() {
    Pthread_mutex_lock(&m);
    while (done == 0)
        Pthread_cond_wait(&c, &m);
    Pthread_mutex_unlock(&m);
}

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;
}
```

Output:

```
parent: begin
child
parent: end
```

Código (link)

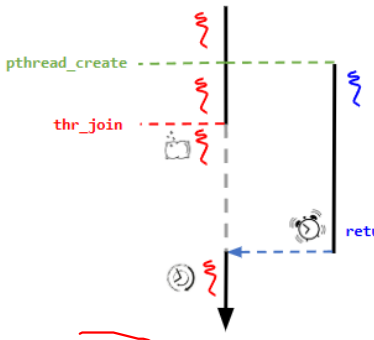




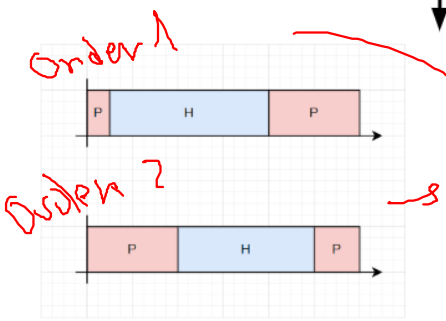
```c
int done = 0;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;

void thr_exit() {
  Pthread_mutex_lock(&m);
  done = 1;
  Pthread_cond_signal(&c);
  Pthread_mutex_unlock(&m);
}

void *child(void *arg) {
  printf("child\n");
  thr_exit();
  return NULL;
}

void thr_join() {
  Pthread_mutex_lock(&m);
  while (done == 0)
    Pthread_cond_wait(&c, &m);
  Pthread_mutex_unlock(&m);
}

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;
}
```

Código (link)

Conclusión



spin lock (desperdicio)




```
parent: begin
child
parent: end
```

¿Como podemos evitar usar el while???

Solucion:

Reimplementacion

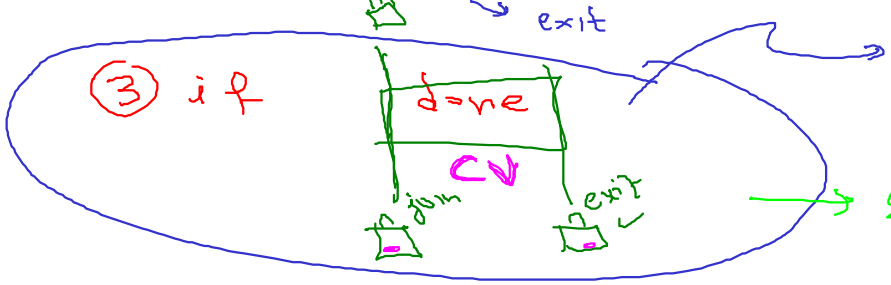
① thr-exit
② thr-join

~~while()~~

5. Reescritura de thr-exit() y thr-join() para no usar while

① ~~while~~ ~~done~~ → no sirve

② if done → no sirve



Función `thr_exit`

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

Función `thr_join`

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