17. Condition Variables

↳ mutex보다 좀 더 abstraction

▼ 어떤 condition이 만족될 때까지 기다리려면?

⇒ condition variable 사용이 유용하다

→ 어떤 condition이 만족되면 process를 깨움 (더 다양한 조건으로 적용 가능)

(spining을 계속 기다리기엔 CPU 낭비가 심하고 부정확할 수 있음)

- example

```c
volatile int done = 0;
void *child(void *arg) {
  printf("child\n");
  done = 1;
  return NULL;
}
int main(int argc, char *argv[]) {
  pthread_t c;
  printf("parent: begin\n");
  pthread_create(&c, NULL, child, NULL); // create child
  while (done == 0); // spin -> 매번 done의 값을 확인해야 함
  printf("parent: end\n");
  return 0;
}
```

done이 1이되면 while을 빠져나가도록 → 비효율적

▼ Condition Variable

특정한 queue라고 생각해보자

↳ thread : 특정 queue에 진입한 뒤 sleep

⇒ state가 변화했을 때 queue에 들어가서 기다리고 있던 다른 thread 깨움

```c
#include <pthread.h>

pthread_cond_wait(); //sleep → mutex 인자 lock해제 + 호출한 thread 재움
pthread_cond_signal(); //wake → thread가 program 내에서 변화가 생겼을 때 깨움.
```

- example

```c
int done = 0;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER; → mutex
pthread_cond_t c = PTHREAD_COND_INITIALIZER; → condition var.

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

int main(int argc, char *argv[]) {
  pthread_t p;
  printf("parent: begin\n");
  pthread_create(&p, NULL, child, NULL);
  thr_join();
  printf("parent: end\n");
  return 0;
}

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

void thr_join() {
  pthread_mutex_lock(&m); (parent)
  while (done == 0)  → main이 잠들게 됨 ⇒ done이 0이되면 빠져나옴.
    pthread_cond_wait(&c, &m); → 해당하는 mutex lock 먼저 호출
  pthread_mutex_unlock(&m);
}
```

critical section

critical section

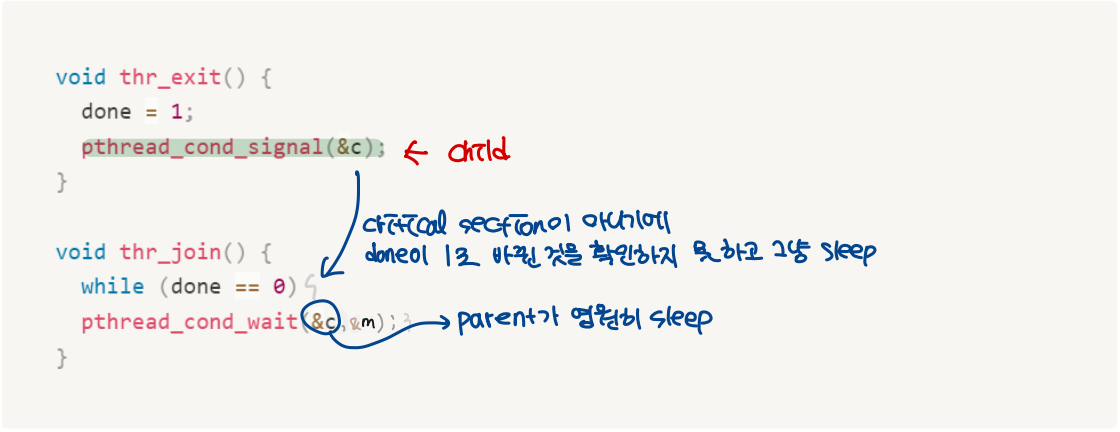
↳ 왜 이렇게 복잡하게 사용? → 하나씩 없을 때의 상황 확인해보자!

①▼ what if → no state variable

- parent가 잠들었을 때 깨워줄 thread가 존재하지 않음(이미 child는 완료)

  ⇒ 계속 sleep

② ▼ **what if → no lock**



- child에 의해 done이 1로 변한 것을 확인하지 못 하고 그냥 sleep → 깨어날 수 x

⇒ mutex lock + condition variable 잘사용해야함.

▼ Producer/Consumer problem → 좀 더 현실적인 예시를 알아보자

- **Producers**
  - generate data items → 크기가 정해져 있는 buffer에 입력
- **Consumers**
  - grab items from buf → cousume them

⇒ buf : shared resource → synchoronization 필요

- examples
  - pipe, web servers

    ↳ grep foo file.txt | wc -l

- example

```
int buffer; // single buffer (queue size = 1)
int count = 0; // initially, empty => (1, 0)
                                      full   empty

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

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

```
//1) if문 example
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) → buf가 꽉 찼을 때 ⇒ 더 생산하면 안 됨.   // p2
      pthread_cond_wait(&cond, &mutex); → 잠시 wait   // p3
    put(i); ⇒ count, buf update                    // p4
    pthread_cond_signal(&cond);                    // p5
    pthread_mutex_unlock(&mutex);                  // p6
  }
}

void *consumer(void *arg) {
  int i;
    for (i = 0; i < loops; i++) {
    pthread_mutex_lock(&mutex);                    // c1
    if (count == 0) → buffer가 비었을 때 ⇒ 팔면 안 됨   // c2
      pthread_cond_wait(&cond, &mutex); → 잠시 wait   // c3
    int tmp = get(); → buffer 값 받아옴              // c4
    pthread_cond_signal(&cond);                    // c5
    pthread_mutex_unlock(&mutex);                  // c6
    printf("%d\n", tmp);
    }
}
```

* 2개 작성하게 되면

⇒ 하나의 producer, consumer에게만 작동함

⇒ consumer thread가 여러개일 때 → while로 확인해야 함.

T_{c1}	State	T_{c2}	State	T_p	State	Count	*Comment*
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_{c1} awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	
	Ready		Ready	p3	Sleep	1	Buffer full; sleep
	Ready	c1	Running		Sleep	1	T_{c2} sneaks in ...
	Ready	c2	Running		Sleep	1	
	Ready	c4	Running		Sleep	0	... and grabs data
	Ready	c5	Running		Ready	0	T_p awoken
	Ready	c6	Running		Ready	0	
c4	Running		Ready		Ready	0	Oh oh! No data

mutex lock
count =0 확인
pthread_cond_wait
(put)
(get)
CPU 자원 어느곳에 줄지 고민
↳ 더이상은 produce X
↳ count가 1일 것이라고 기대 ⇒ 비정상 종료

wake된 thread(T_c1)이 실행될 때 state 유지될 것이라는 보장 x
↳ 이 사이에 다른 consummer (T_c2가 실행될 수 있음)

```
//2) while문 example

cond_t cond;
mutex_t mutex;
void *producer(void *arg) {
  int i;
  for (i = 0; i < loops; i++) {
    pthread_mutex_lock(&mutex);
    while (count == 1)
      pthread_cond_wait(&cond, &mutex);
    put(i);
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
  }
}
//같은 condition var에 consumer, producer 모두 들어가 있음
⇒동시에 ready일 때 signal 오면 무엇을 실행할 지 x

void *consumer(void *arg) {
int i;
  for (i = 0; i < loops; i++) {
    pthread_mutex_lock(&mutex);
    while (count == 0)
      pthread_cond_wait(&cond, &mutex);
    int tmp = get();
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
    printf("%d\n", tmp);
  }
}
```

T_{c1}	State	T_{c2}	State	T_p	State	Count	Comment
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	→ 구매할 물건이 X
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0 (put)	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_{c1} awoken
	Ready		Sleep	p6	Running	1	
	Ready		Sleep	p1	Running	1	
	Ready		Sleep	p2	Running	1	
	Ready		Sleep	p3	Sleep	1	Must sleep (full)
c2	Running		Sleep		Sleep	1	Recheck condition
c4	Running		Sleep		Sleep	0	T_{c1} grabs data
c5	Running		Ready		Sleep	0	Oops! Woke T_{c2}
c6	Running		Ready		Sleep	0	
c1	Running		Ready		Sleep	0	
c2	Running		Ready		Sleep	0	
c3	Sleep		Ready		Sleep	0	Nothing to get
	Sleep	c2	Running		Sleep	0	
	Sleep	c3	Sleep		Sleep	0	Everyone asleep...

다른 consumer thread는 깨우지 못 하고 prducer만 깨울 수 있음 → CV 두 개 사용?..

↳ But, 같은 condition variable에 있기때문에 먼저 ready 한 놈이 실행됨. ⇒ consumer 실행

↙ 해결!

```
//3) while & Two CVs
cond_t empty, fill;
mutex_t mutex;

void *producer(void *arg) {
  int i;
  for (i = 0; i < loops; i++) {
    pthread_mutex_lock(&mutex);
    while (count == 1)
      pthread_cond_wait(&empty, &mutex);
    put(i);
    pthread_cond_signal(&fill);
    pthread_mutex_unlock(&mutex);
  }
}

void *consumer(void *arg) {
  int i;
  for (i = 0; i < loops; i++) {
    pthread_mutex_lock(&mutex);
    while (count == 0)
      pthread_cond_wait(&fill, &mutex);
    int tmp = get();
    pthread_cond_signal(&empty);
    pthread_mutex_unlock(&mutex);
    printf("%d\n", tmp);
  }
}
```

↕ Producer랑 consumer가 다른 condition variable 사용

▼ More Concurrency & Efficiency

```
int buffer[MAX];
int fill_ptr = 0;
int use_ptr = 0;
int count = 0;

void put(int value) {
  buffer[fill_ptr] = value;
  fill_ptr = (fill_ptr + 1) % MAX;
  count++;
}

int get() {
  int tmp = buffer[use_ptr];
  use_ptr = (use_ptr + 1) % MAX;
  count--;
  return tmp;
}

cond_t empty, fill;
mutex_t mutex;

void *producer(void *arg) {
  int i;
  for (i = 0; i < loops; i++) {
    pthread_mutex_lock(&mutex);
    while (count == MAX)  → buf가 끝까지 차면
    pthread_cond_wait(&empty, &mutex);
    put(i);
    pthread_cond_signal(&fill);
    pthread_mutex_unlock(&mutex);
  }
}

void *consumer(void *arg) {
  int i, tmp;
  for (i = 0; i < loops; i++) {
    pthread_mutex_lock(&mutex);
    while (count == 0) → buf가 모두 비었다면
      pthread_cond_wait(&fill, &mutex);
    tmp = get();
    pthread_cond_signal(&empty);
    pthread_mutex_unlock(&mutex);
    printf("%d\n", tmp);
  }
}
```

- Covering Conditions
  - thread가 wake할 필요가 있는 경우 모두 cover
  - *pthread_cond_broadcast();*
    - 기다리는 모든 thread 깨우기

↳ 인자로 전달된 condition variable의 모든 thread 깨움
⇒ 들어갈만한 state를 가진 thread 찾음