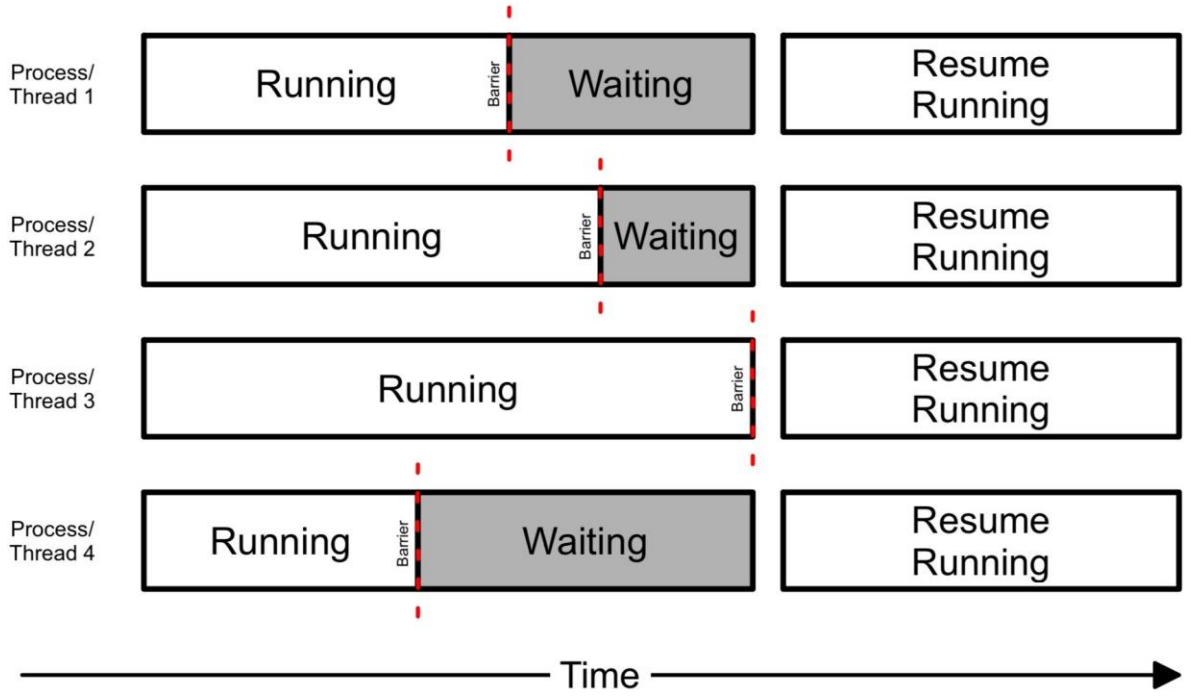
CS214-system programming

- Section 03/08 recitation 9

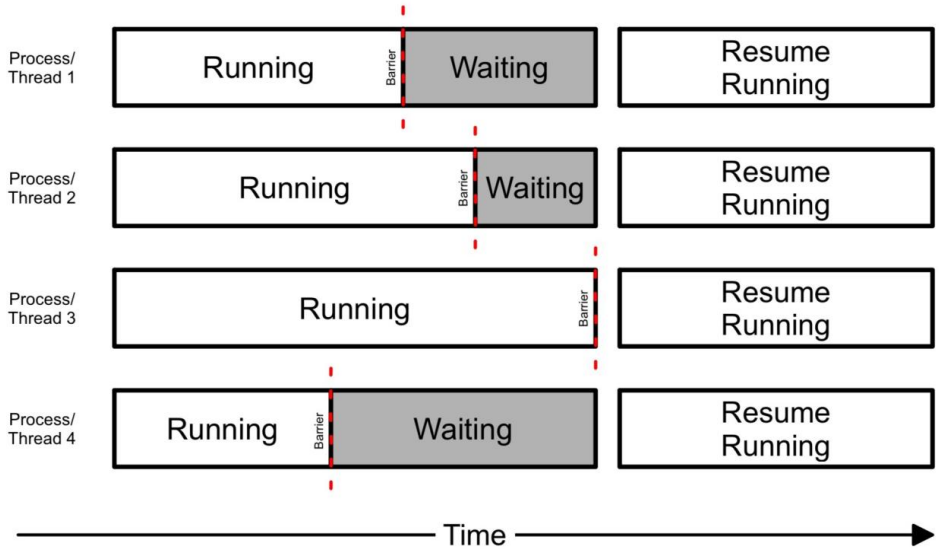
Yunhe Gao
yg397@scarletmail.rutgers.edu

Barrier

- Suppose that we want threads to continue execution only after some other threads of interest reach some stage. That is, the threads will only continue to execute after they all reach some pre-communicated stages.
- A **barrier** is a synchronization method that allows this.
- When a thread reaches a barrier, it will wait at the barrier until all the other threads reach the barrier. They will continue to execute once they have all reached the barrier.



Barrier



https://medium.com/@jaydesai36/barrier-synchronization-in-threads-3c56f947047

Syntax :

```
int pthread_barrier_init(pthread_barrier_t * barrier,
     const pthread_barrierattr_t * attr, unsigned count);
int pthread_barrier_destroy(pthread_barrier_t * barrier);
int pthread_barrier_wait(pthread_barrier_t * barrier);
```

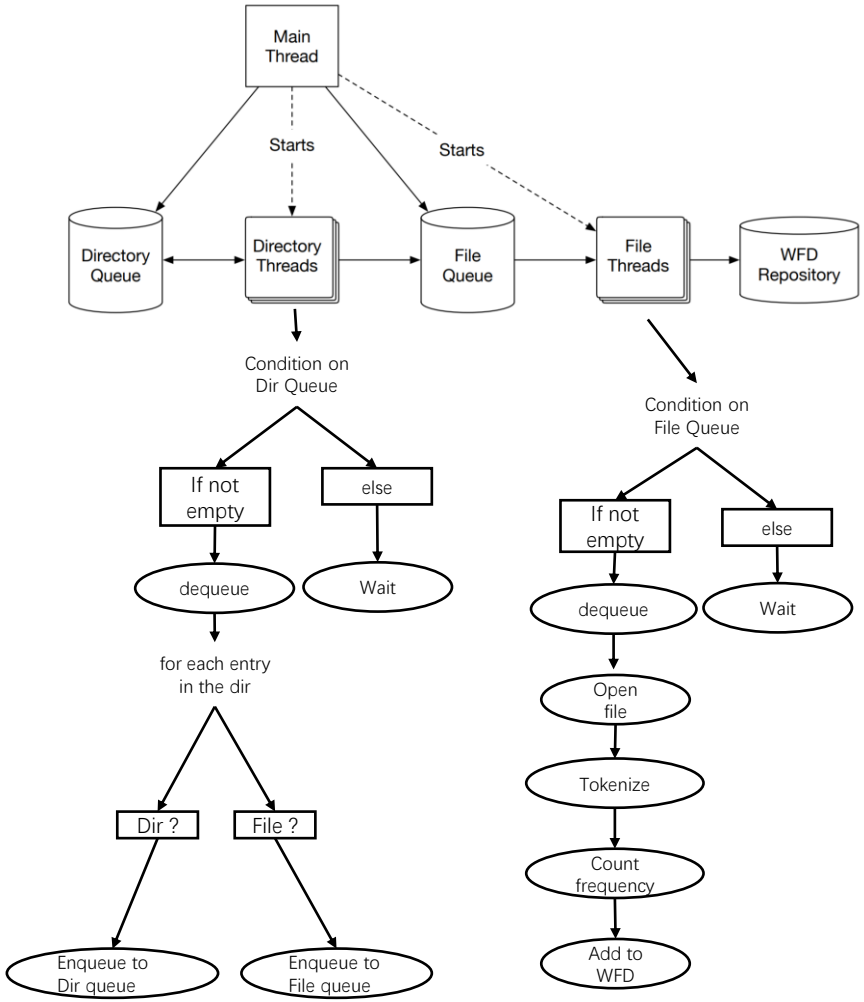
Return Value : On success **_init()** & **_destroy()** return 0 otherwise error number.(EAGAIN-system lacks the necessary resources to initialize another barrier, EINVAL-value specified by ***count*** is equal to zero, ENOMEM-insufficient memory exists to initialize the barrier)

Description : The **pthread_barrier_init()** allocate resources required to use the barrier referenced by ***barrier*** and initialize the ***barrier*** with attribute referenced by ***attr***. If ***attr*** is NULL, the default barrier attributes applied to ***barrier***. The ***count*** specifies the number of threads that must call **pthread_barrier_wait()**. The ***count*** must be greater than zero.

The **pthread_barrier_destroy()** destroy the barrier referenced by ***barrier*** and release any resources used by the barrier.

The **pthread_barrier_wait()** synchronize participating threads at the barrier referenced by ***barrier***. The calling thread block untill the required number of threads(mentioned in ***count*** while initilizing ***barrier***) have called **pthread_barrier_wait()** specifying the ***barrier***. When the required number of threads have called **pthread_barrier_wait()** specifying the barrier, the constant PTHREAD_BARRIER_SERIAL_THREAD shall be returned to one unspecified thread and zero shall be returned to each remaining threads.

Project


Main Thread
Starts
Starts
Directory Queue
Directory Threads
File Queue
File Threads
WFD Repository
Condition on Dir Queue
If not empty
else
dequeue
Wait
for each entry in the dir
Dir ?
File ?
Enqueue to Dir queue
Enqueue to File queue
Condition on File Queue
If not empty
else
dequeue
Wait
Open file
Tokenize
Count frequency
Add to WFD

Synchronized Queue

- The enqueue and dequeue operation are not atomic
- We don't know when the thread switching happens
- We need to make sure enqueue and dequeue to be thread safe
- Refer to queue.c and queue2.c

```
typedef struct {
        int data[QSIZE];
        unsigned count;
        unsigned head;
        pthread_mutex_t lock;
        pthread_cond_t read_ready;
        pthread_cond_t write_ready;
} queue_t;
```

```
// add item to end of queue
// if the queue is full, block until space becomes available
int enqueue(queue_t *Q, int item)
{
        pthread_mutex_lock(&Q->lock);

        while (Q->count == QSIZE) {
                pthread_cond_wait(&Q->write_ready, &Q->lock);
        }

        unsigned i = Q->head + Q->count;
        if (i >= QSIZE) i -= QSIZE;

        Q->data[i] = item;
        ++Q->count;

        pthread_cond_signal(&Q->read_ready);

        pthread_mutex_unlock(&Q->lock);

        return 0;
}
```

```
int dequeue(queue_t *Q, int *item)
{
        pthread_mutex_lock(&Q->lock);

        while (Q->count == 0) {
                pthread_cond_wait(&Q->read_ready, &Q->lock);
        }

        *item = Q->data[Q->head];
        --Q->count;
        ++Q->head;
        if (Q->head == QSIZE) Q->head = 0;

        pthread_cond_signal(&Q->write_ready);

        pthread_mutex_unlock(&Q->lock);

        return 0;
}
```

File Handling

Example:

b.txt has the following sentence:

No pains, no gains.

no: 2 times

pains: 1 times

gains: 1 times

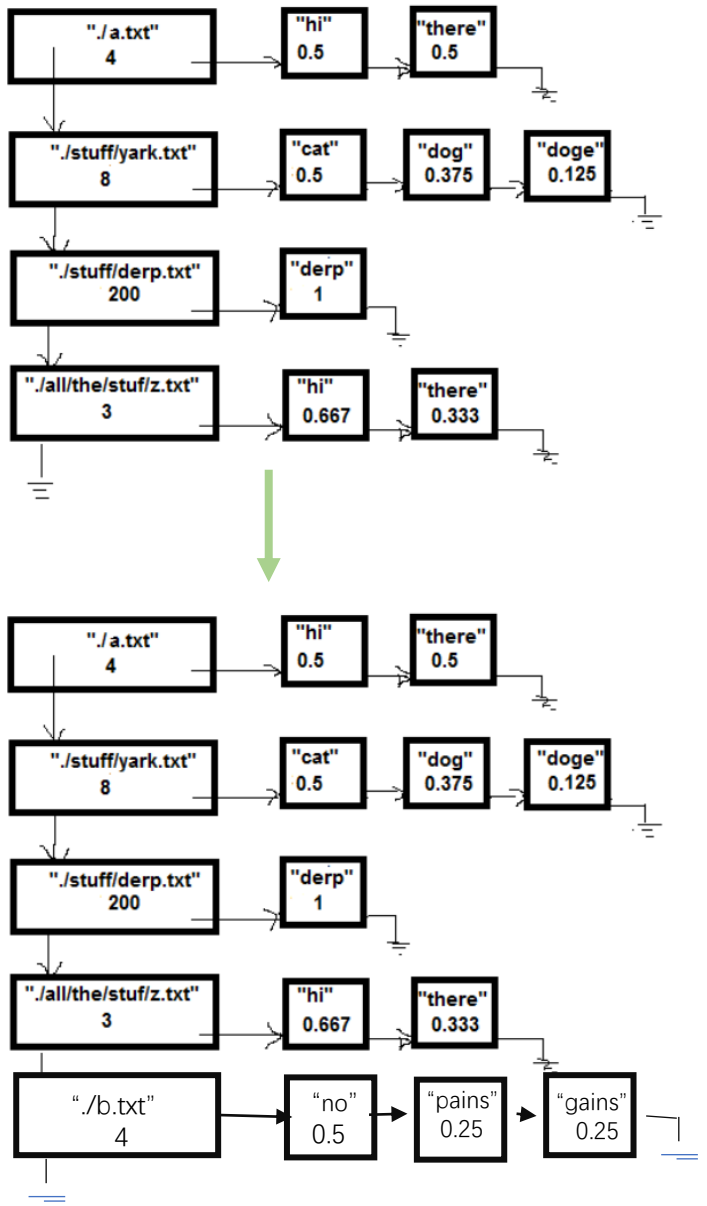
total 4 tokens

Thus, the frequency are:

no: 0.5

pains: 0.25

gains: 0.25



Compute JSD

(a) F_1

hi	0.5
there	0.5

(b) F_2

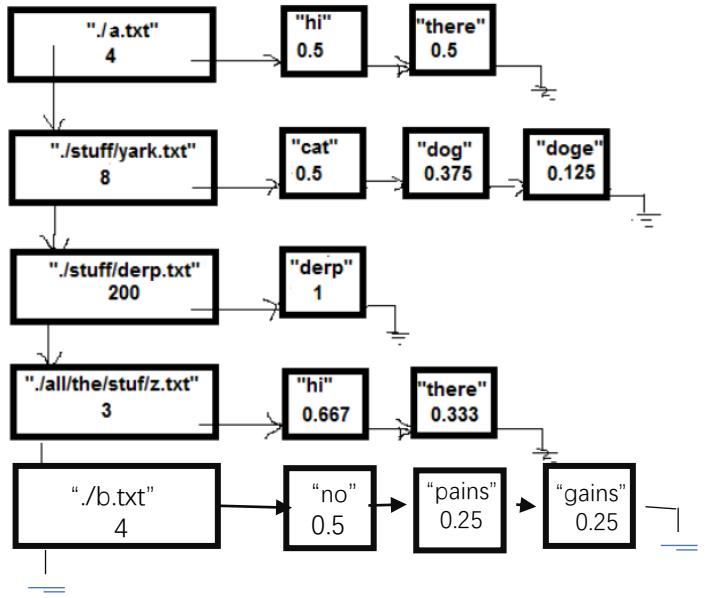
hi	0.5
out	0.25
there	0.25

(c) $\overline{F}$

hi	0.5
out	0.125
there	0.375

$$
\begin{aligned}
KLD(F_1||\overline{F}) &= 0.5 \cdot \log_2\left(\frac{0.5}{0.5}\right) + 0.5 \cdot \log_2\left(\frac{0.5}{0.375}\right) \\
&\approx 0.5 \cdot 0 + 0.5 \cdot 0.415 \\
&\approx 0.2075 \\
KLD(F_2||\overline{F}) &= 0.5 \cdot \log_2\left(\frac{0.5}{0.5}\right) + 0.25 \cdot \log_2\left(\frac{0.25}{0.125}\right) + 0.25 \cdot \log_2\left(\frac{0.25}{0.375}\right) \\
&\approx 0.5 \cdot 0 + 0.25 \cdot 1 + 0.25 \cdot -0.585 \\
&\approx 0.1038 \\
JSD(F_1||F_2) &\approx \sqrt{\tfrac{1}{2}0.2075 + \tfrac{1}{2}0.1038} \\
&\approx 0.3945
\end{aligned}
$$

Figure 1: Computing the JSD for two files



Thanks