OS Assignment 8 Report

name：刘乐奇

sid：12011327

Ubuntu用户名：lynchrocket

1. I/O

(1) What are the pros and cons of polling and interrupt-based I/O?

	pros	cons
polling	simple & practical	inefficient & inconvenient
interrupt	no spinning during I/O	context switch costs

(2) What are the differences between PIO and DMA?

	PIO	DMA
need CPU ?	CPU controls	no need of CPU
transfer speed	slower	faster
devices used for	slower devices	high-speed devices
who does data copy ?	CPU	DMA controller

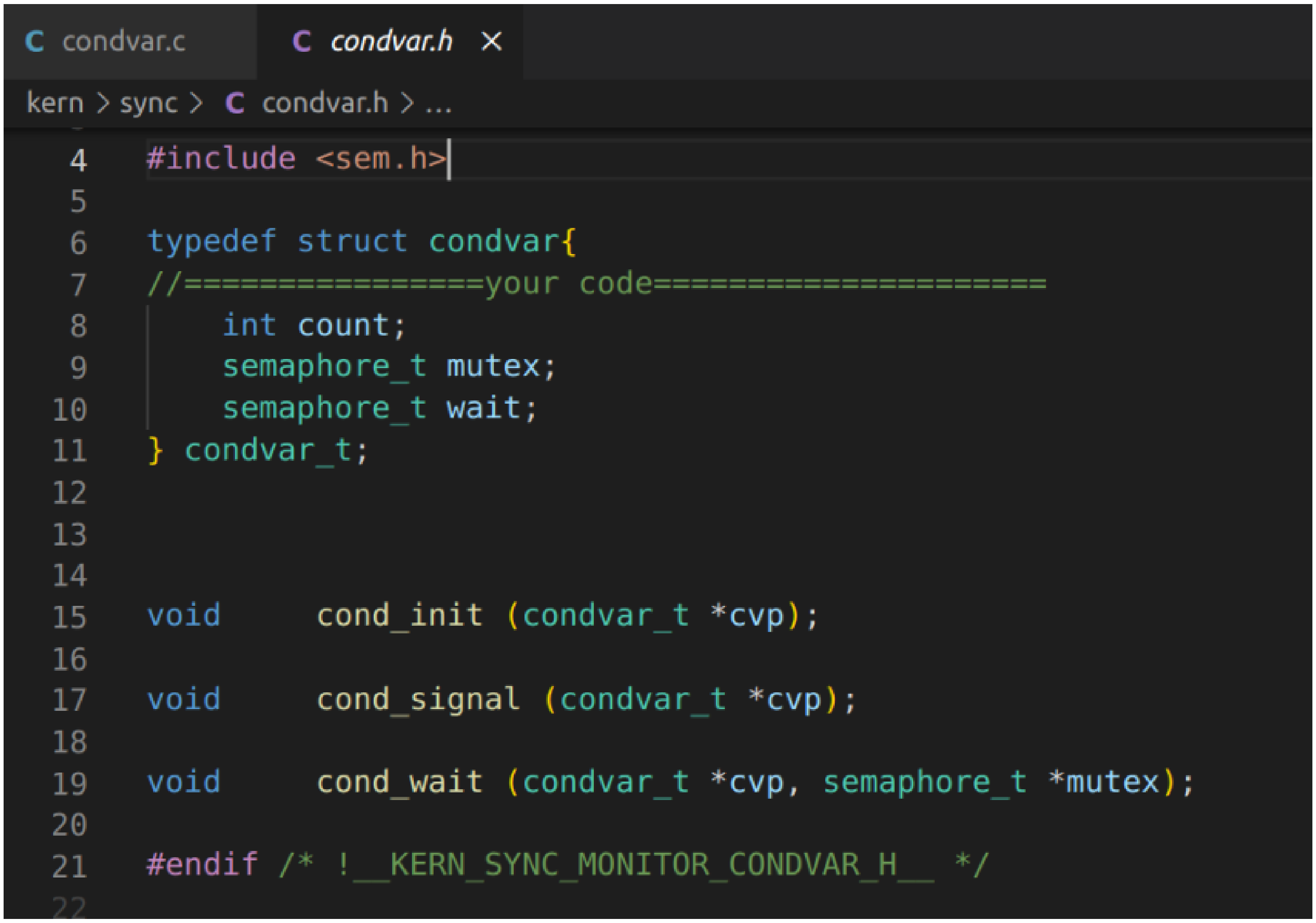
(3) How to protect memory-mapped I/O and explicit I/O instructions from being abused by malicious user process?

1. Priviledge levels: Restrict access to memory-mapped I/O and explicit I/O instructions to trusted processes running at higher privilege levels.
2. Memory protection：Maintain a table that maps I/O instructions to specific devices and only allow authorized processes to use these instructions.

2. Condition variable

The definition of condition variable.

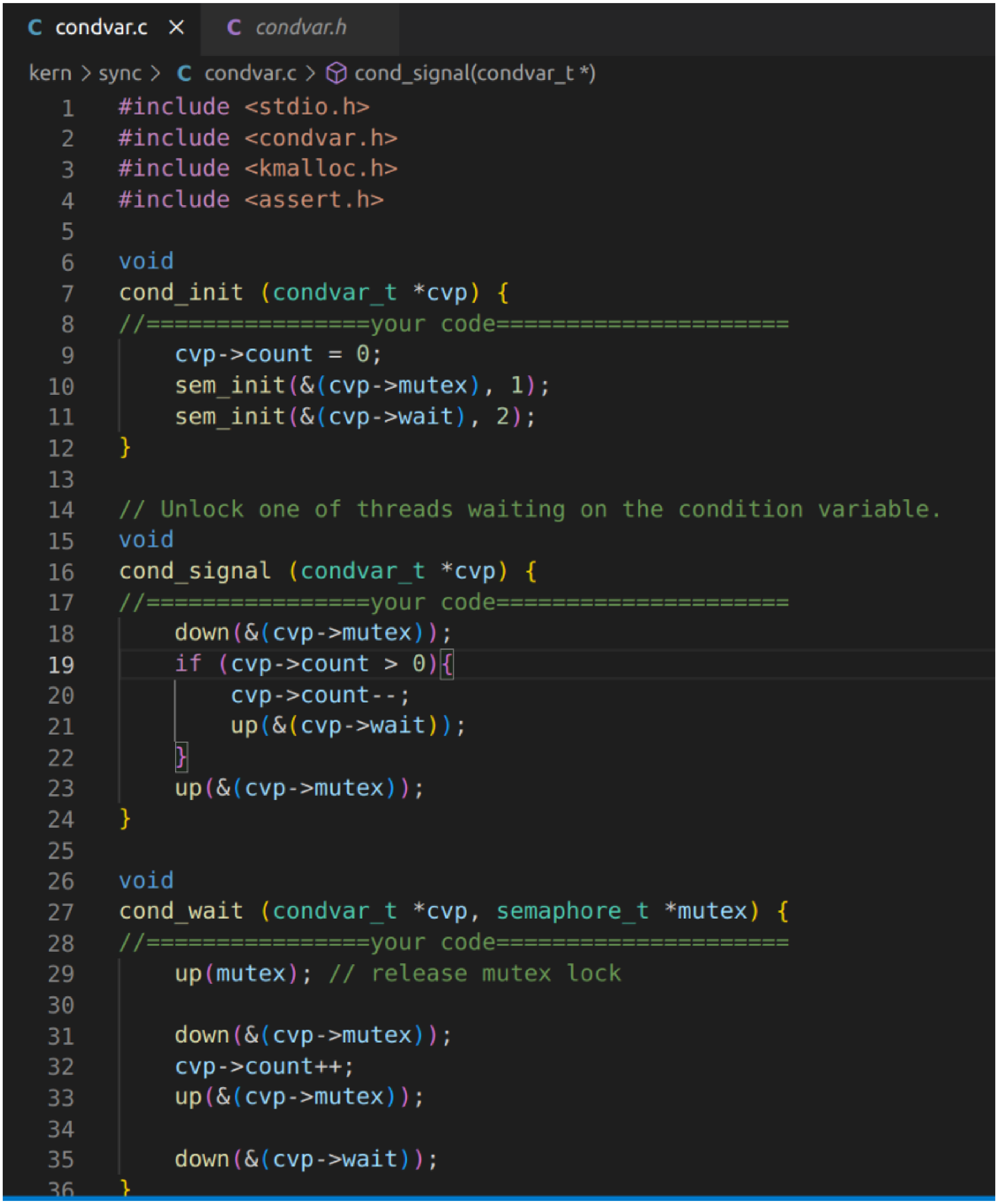
name	description
count	the number of threads waiting on the conditional variable
mutex	a mutex lock used to protect the conditional variable, ensuring that only one thread can modify the state of the conditional variable at any given time
wait	a semaphore used to wait for the conditional variable, indicating how many threads are waiting for the state of the conditional variable to change



```c
#include <sem.h>

typedef struct condvar{
//===============your code=====================
    int count;
    semaphore_t mutex;
    semaphore_t wait;
} condvar_t;



void     cond_init (condvar_t *cvp);

void     cond_signal (condvar_t *cvp);

void     cond_wait (condvar_t *cvp, semaphore_t *mutex);

#endif /* !__KERN_SYNC_MONITOR_CONDVAR_H__ */
```

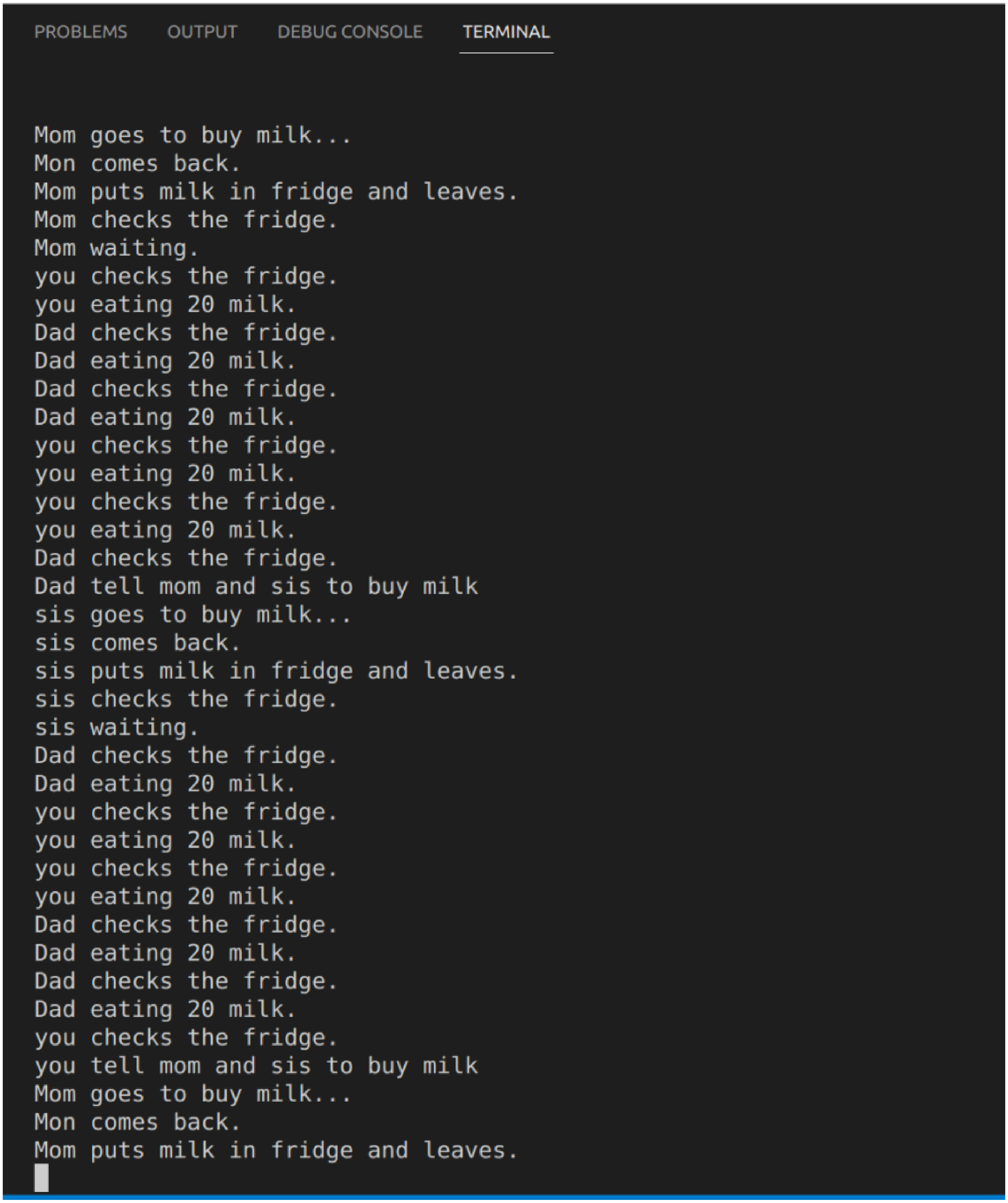
The implement of functions.

The condition variable needs to be initialized in `cond_init` . When invoking `cond_wait` , the mutex lock should be released first in case that other threads are blocked due to have not obtained the mutex lock. Then increase the `count` of condition variable by `1` between a mutex lock in case other threads modify the `count` concurrently, and be waiting. When invoking `cond_signal`, it will check if there is any thread waiting between a mutex lock. If yes, then decrease the `count` of condition variable by `1` and notify the waiting thread.



```c
#include <stdio.h>
#include <condvar.h>
#include <kmalloc.h>
#include <assert.h>

void
cond_init (condvar_t *cvp) {
//================your code=====================
    cvp->count = 0;
    sem_init(&(cvp->mutex), 1);
    sem_init(&(cvp->wait), 2);
}

// Unlock one of threads waiting on the condition variable.
void
cond_signal (condvar_t *cvp) {
//================your code=====================
    down(&(cvp->mutex));
    if (cvp->count > 0){
        cvp->count--;
        up(&(cvp->wait));
    }
    up(&(cvp->mutex));
}

void
cond_wait (condvar_t *cvp, semaphore_t *mutex) {
//================your code=====================
    up(mutex); // release mutex lock

    down(&(cvp->mutex));
    cvp->count++;
    up(&(cvp->mutex));

    down(&(cvp->wait));
}
```

The running result is as below.



```
Mom goes to buy milk...
Mon comes back.
Mom puts milk in fridge and leaves.
Mom checks the fridge.
Mom waiting.
you checks the fridge.
you eating 20 milk.
Dad checks the fridge.
Dad eating 20 milk.
Dad checks the fridge.
Dad eating 20 milk.
you checks the fridge.
you eating 20 milk.
you checks the fridge.
you eating 20 milk.
Dad checks the fridge.
Dad tell mom and sis to buy milk
sis goes to buy milk...
sis comes back.
sis puts milk in fridge and leaves.
sis checks the fridge.
sis waiting.
Dad checks the fridge.
Dad eating 20 milk.
you checks the fridge.
you eating 20 milk.
you checks the fridge.
you eating 20 milk.
Dad checks the fridge.
Dad eating 20 milk.
Dad checks the fridge.
Dad eating 20 milk.
you checks the fridge.
you tell mom and sis to buy milk
Mom goes to buy milk...
Mon comes back.
Mom puts milk in fridge and leaves.
```