CSCI4430 Tutorial – 5
Assignment 2 Review(part 2)

(slides adapted from csci4430@2018spring by Xiaolu Li)

zncheng@cse.cuhk.edu.hk

Feb 28, 2019

Last Week

- Application Overview
- Lossy network Overview
- Selective Repeat Overview

This Week

- Call Back
  - Mutual Exclusions
  - Condition Variables
- Implementation Hints
  - MYSR_Packet
  - Window is full?
  - Timer
  - Terminate
  - multi-threading
- Other Issues

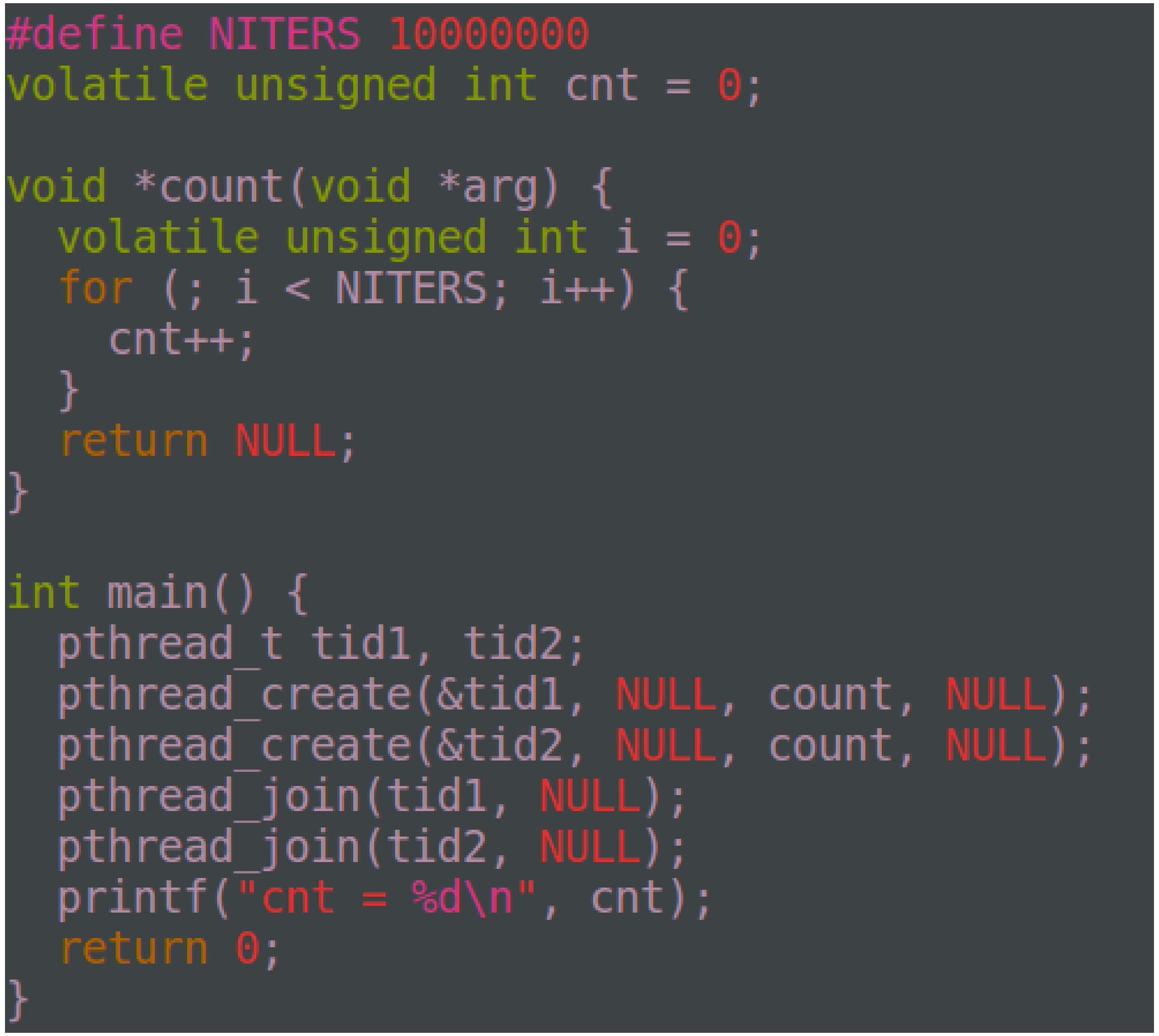
Call Back: Mutual Exclusions

- Mutex protects shared resources
  - pthread_mutex_lock(): acquires a mutex lock, or wait until the lock is released by some others.
  - pthread_mutex_unlock(): releases the mutex lock.

Call Back: Mutual Exclusions

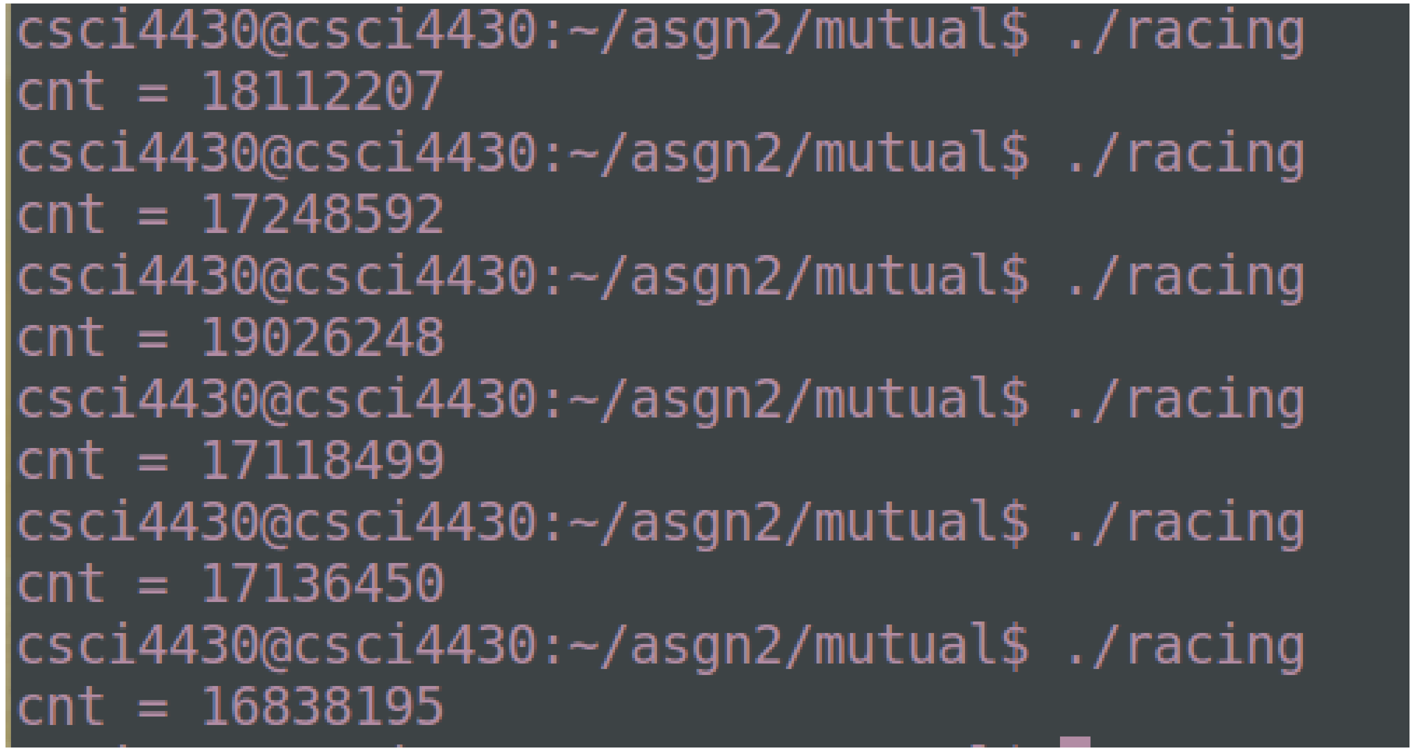
- We provide mutual.tar.gz in tutorial website
- Download it and run `make` to compile the example source code
- Example code
  - There is an global integer cnt initialized to 0;
  - We create two threads, each of them want to add 1 to cnt by 10000000 times.
- racing.c: source code does not use mutex.
  - The final result of cnt is less than 20000000.
- avoid_racing.c: source code uses mutex to avoid racing.
  - The final result of cnt is equals to 20000000.

racing.c

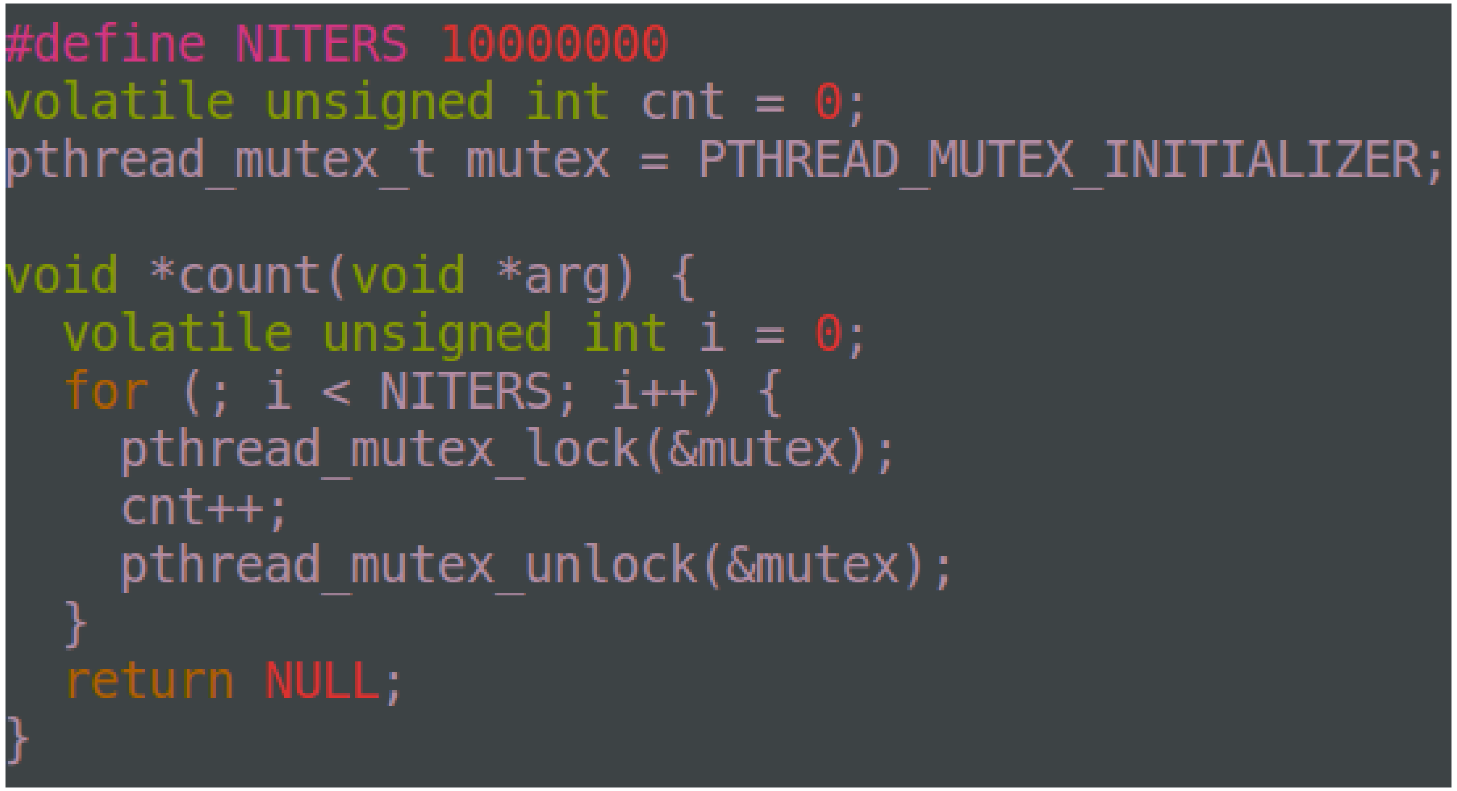


```c
#define NITERS 10000000
volatile unsigned int cnt = 0;

void *count(void *arg) {
  volatile unsigned int i = 0;
  for (; i < NITERS; i++) {
    cnt++;
  }
  return NULL;
}

int main() {
  pthread_t tid1, tid2;
  pthread_create(&tid1, NULL, count, NULL);
  pthread_create(&tid2, NULL, count, NULL);
  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);
  printf("cnt = %d\n", cnt);
  return 0;
}
```

racing.c: result

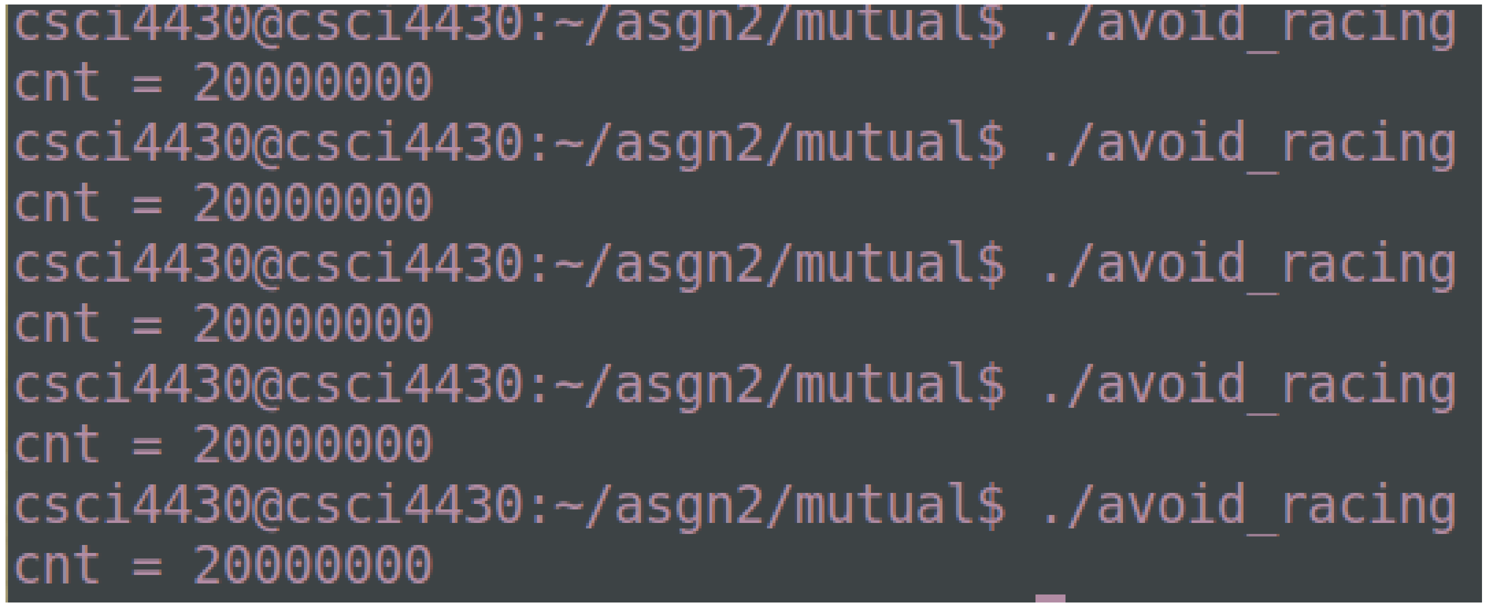

```
csci4430@csci4430:~/asgn2/mutual$ ./racing
cnt = 18112207
csci4430@csci4430:~/asgn2/mutual$ ./racing
cnt = 17248592
csci4430@csci4430:~/asgn2/mutual$ ./racing
cnt = 19026248
csci4430@csci4430:~/asgn2/mutual$ ./racing
cnt = 17118499
csci4430@csci4430:~/asgn2/mutual$ ./racing
cnt = 17136450
csci4430@csci4430:~/asgn2/mutual$ ./racing
cnt = 16838195
```

avoid_racing.c



```c
#define NITERS 10000000
volatile unsigned int cnt = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *count(void *arg) {
  volatile unsigned int i = 0;
  for (; i < NITERS; i++) {
    pthread_mutex_lock(&mutex);
    cnt++;
    pthread_mutex_unlock(&mutex);
  }
  return NULL;
}
```

avoid_racing.c: result



```
csci4430@csci4430:~/asgn2/mutual$ ./avoid_racing
cnt = 20000000
csci4430@csci4430:~/asgn2/mutual$ ./avoid_racing
cnt = 20000000
csci4430@csci4430:~/asgn2/mutual$ ./avoid_racing
cnt = 20000000
csci4430@csci4430:~/asgn2/mutual$ ./avoid_racing
cnt = 20000000
csci4430@csci4430:~/asgn2/mutual$ ./avoid_racing
cnt = 20000000
```

Call Back: Condition Variables

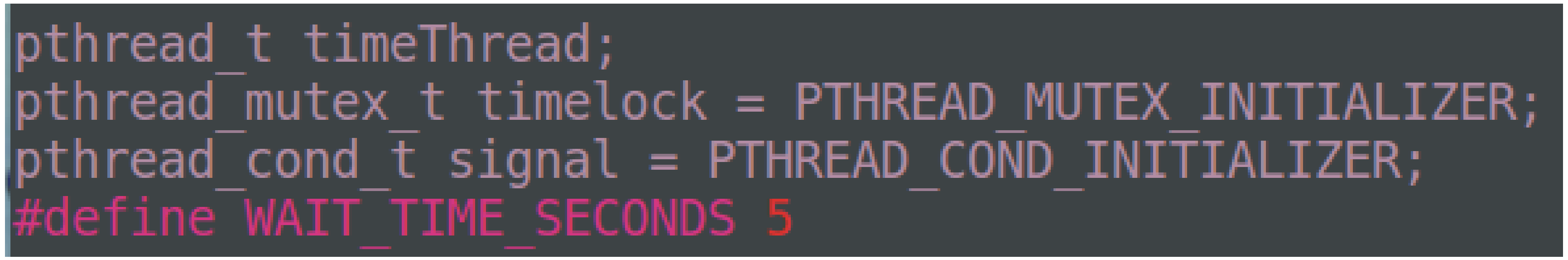
- Condition variables let threads to sleep waiting for some conditions to occur.
  - pthread_cond_wait(): unlocks the mutex, waits on the condition until a signal is received, and re-acquires the lock
  - pthread_cond_timedwait(): unlocks the mutex and waits on the condition until a signal is received or the timeout period expires
  - pthread_cond_signal(): restarts one waiting thread

Call Back: Condition Variables

- We provide cond.tar.gz in tutorial website
- Download it and run `make` to compile the source code
- source code
  - We start a thread to wait for the signal within 5 seconds.
  - We type “signal” from stdin to trigger the signal.

cond.c

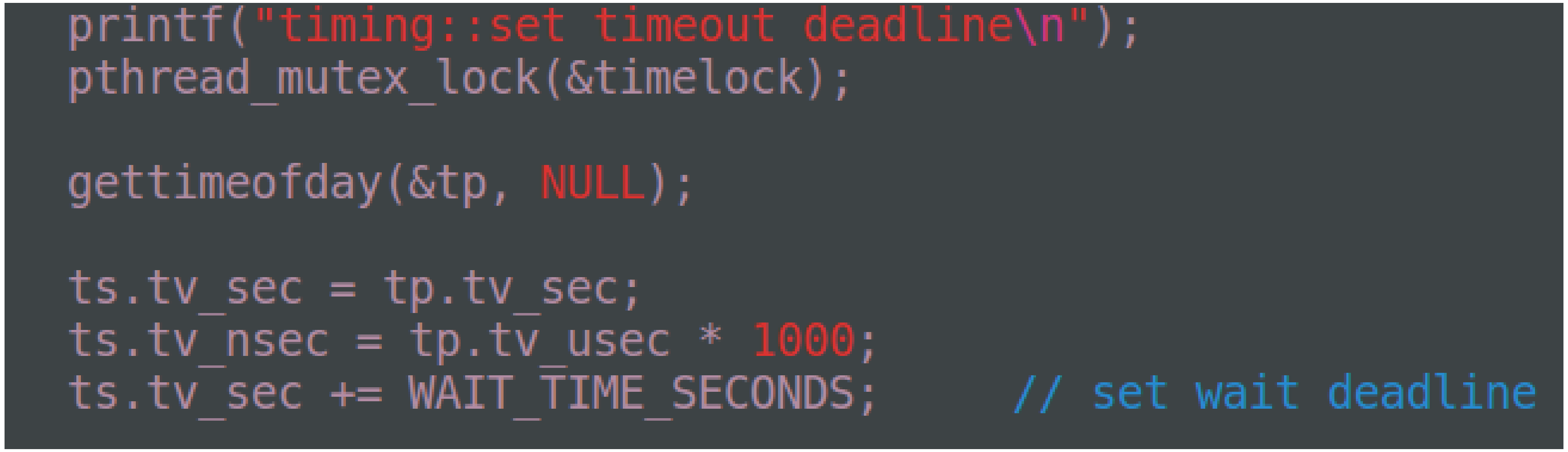
- variables used



```
pthread_t timeThread;
pthread_mutex_t timelock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t signal = PTHREAD_COND_INITIALIZER;
#define WAIT_TIME_SECONDS 5
```

cond.c: timing thread

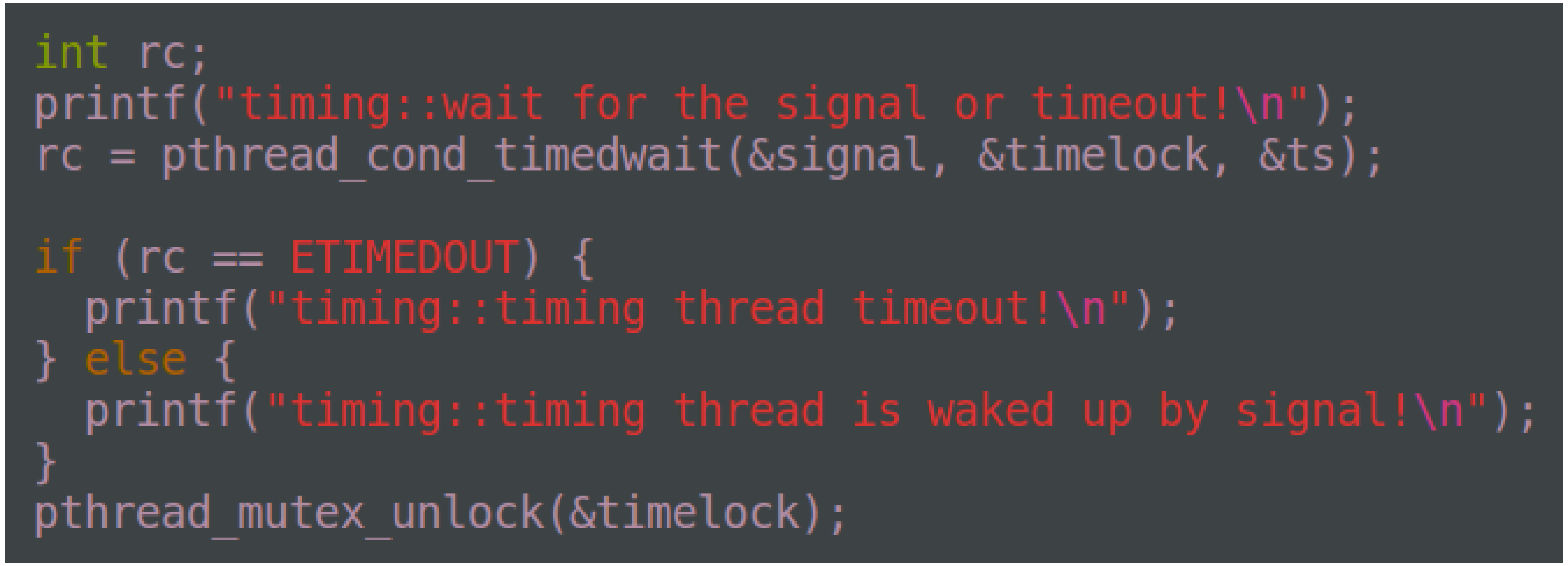
- set timeout deadline



```c
printf("timing::set timeout deadline\n");
pthread_mutex_lock(&timelock);

gettimeofday(&tp, NULL);

ts.tv_sec = tp.tv_sec;
ts.tv_nsec = tp.tv_usec * 1000;
ts.tv_sec += WAIT_TIME_SECONDS;     // set wait deadline
```

cond.c: timing thread

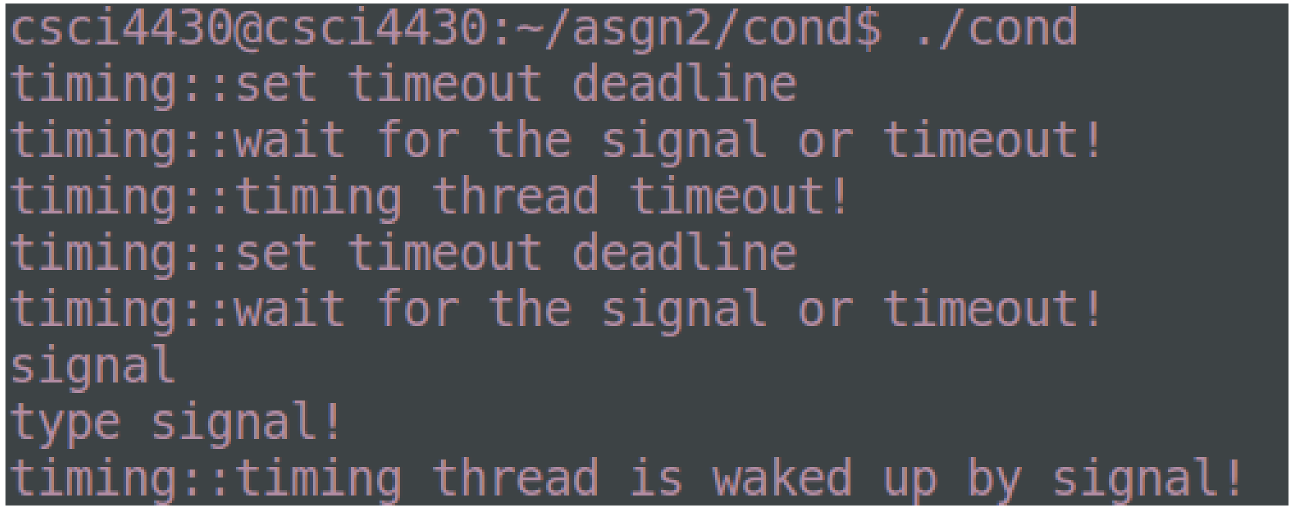
- pthread_cond_timedwait



```
int rc;
printf("timing::wait for the signal or timeout!\n");
rc = pthread_cond_timedwait(&signal, &timelock, &ts);

if (rc == ETIMEDOUT) {
  printf("timing::timing thread timeout!\n");
} else {
  printf("timing::timing thread is waked up by signal!\n");
}
pthread_mutex_unlock(&timelock);
```

Call Back: Condition Variables

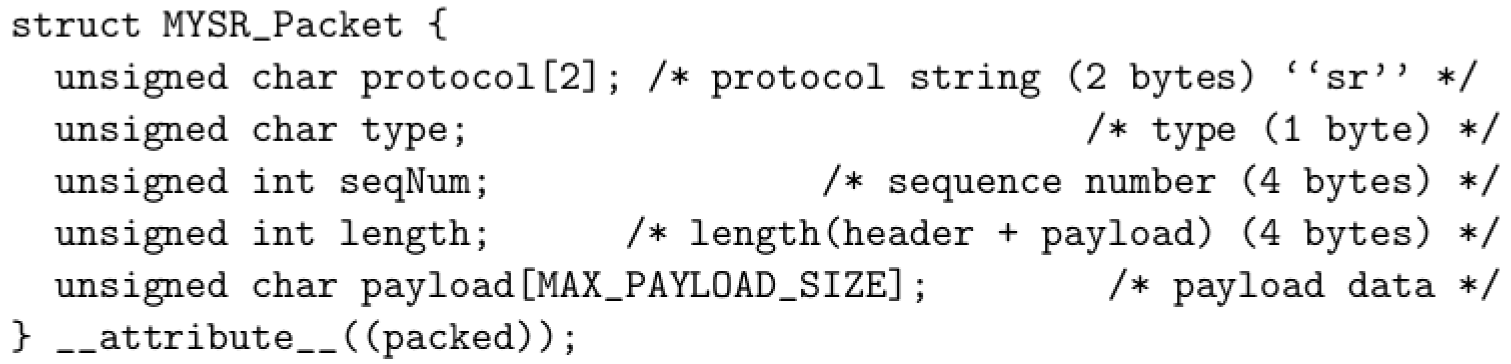
- At first, we do not type anything
  - The thread which waits for the signal will timeout and goes to another round to wait for the signal
- Then we type “signal”
  - The thread which waits for the signal will be waked up



```
csci4430@csci4430:~/asgn2/cond$ ./cond
timing::set timeout deadline
timing::wait for the signal or timeout!
timing::timing thread timeout!
timing::set timeout deadline
timing::wait for the signal or timeout!
signal
type signal!
timing::timing thread is waked up by signal!
```

Implementation Hints: MYSR_Packet

- The basic unit in Selective Repeat layer is MYSR_Packet



```
struct MYSR_Packet {
  unsigned char protocol[2]; /* protocol string (2 bytes) ‘‘sr’’ */
  unsigned char type;                              /* type (1 byte) */
  unsigned int seqNum;                  /* sequence number (4 bytes) */
  unsigned int length;      /* length(header + payload) (4 bytes) */
  unsigned char payload[MAX_PAYLOAD_SIZE];        /* payload data */
} __attribute__((packed));
```

- protocol: “sr”
- type
- seqNum
- length: the whole packet length(2+1+4+4+payload len)
- payload: application data

Implementation Hints: MYGBN_Packet

- There are 3 types of MYSR_Packet
  - DataPacket
  - AckPacket
  - EndPacket

Implementation Hints: Data_Packet

- DataPacket
  - As we discussed last week, when new data comes, the sender first check whether the data can be sent. If it can be sent immediately, the data will be wrapped into DataPacket and the DataPacket will be assigned a seqNum.
  - We should be careful to the payload size. The maximum payload size of the DataPacket is 512 bytes.
  - When the size of application data is larger than the maximum payload size, we may need to partition it first.

Implementation Hints: AckPacket

- AckPacket
  - When receiver returns ACK to the sender, it is wrapped into an AckPacket
  - AckPacket does not have payload.

Implementation Hints: EndPacket

- EndPacket
  - Sender sends EndPacket to receiver to tell that the sender wants to terminate the data transfer.
  - We will discuss it later.

Implementation Hints: Window is full?

- At sender side, what if new data comes but the current window is full of un-acked packets?
  - create a queue to cache the new coming data?
    - size of the queue?
  - use mutex and condition variables?
    - let the application wait until there is vacancy in current window?
    - implement a blocking queue?
- It’s an implementation issue and you are free to design your own method.
- Please do not just refuse the request and keeps application checking whether it can send data!

Implementation Hints: Timer

- In our assignment, you may need to set a timer for each individual packet.
  - When a new packet comes and the window is available
    - We start a timer for the packet.
    - Timer wait for a signal to reset, or stop.
  - When sender receives an ACK for a packet in current window.
    - the signal should issued and the timer should be stopped.
    - If the ACK packet is the oldest unACKed packet in the window, we may also slide the window.
  - When there is timeout event
    - Resend the corresponding packet of this timer
    - Timer should be reset.

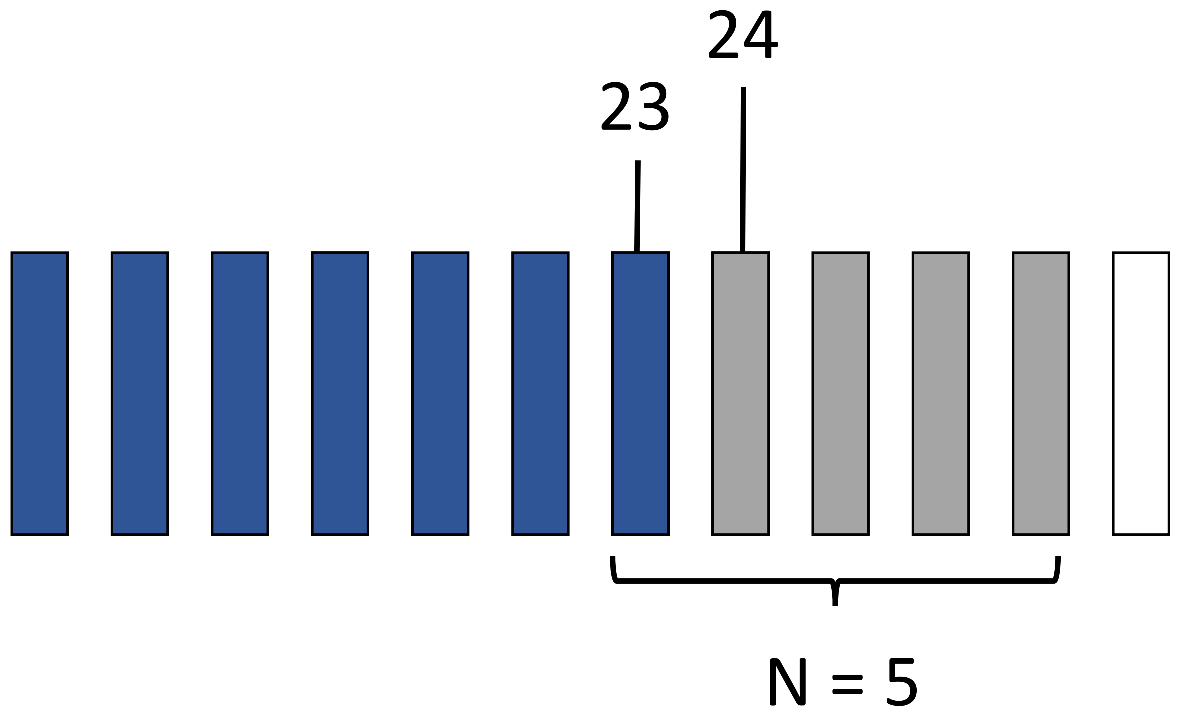
Implementation Hints: Timer

- We may need a condition variable and a mutex to control the waiting within a timeout period.
- We may also need another variable to indicate whether the timer should be started or stopped.

Implementation Hints: Terminate

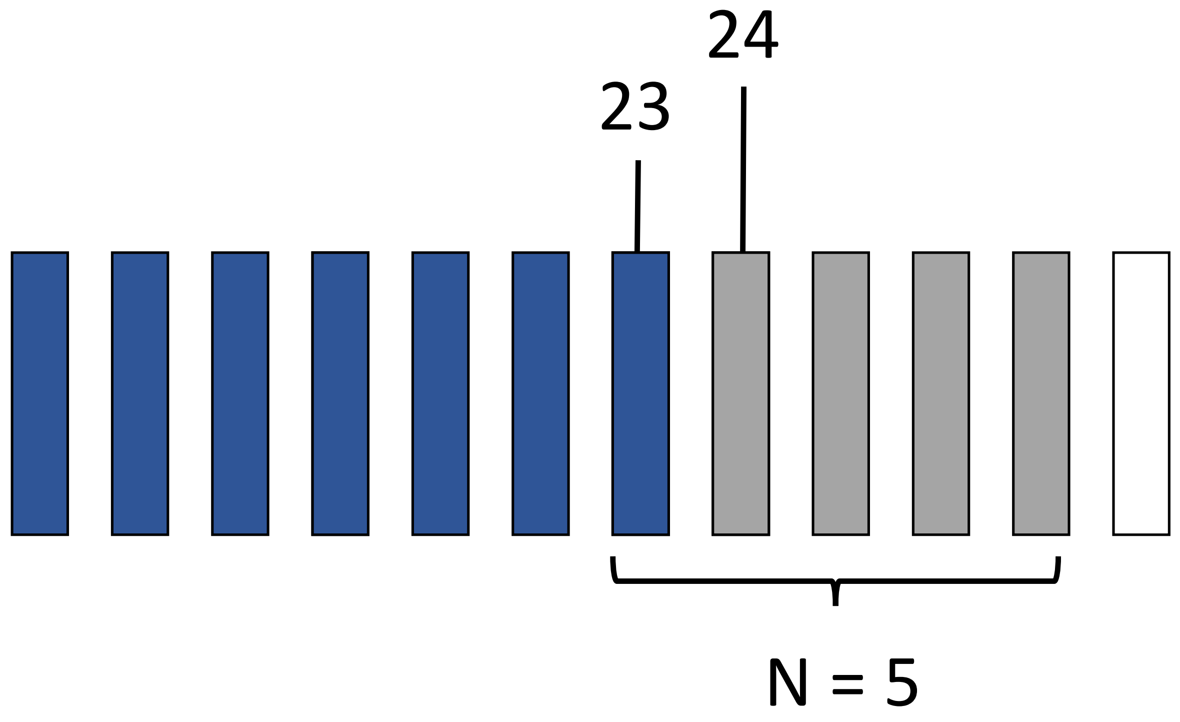
- myftpclient is expected to terminate gracefully after it sends out all the data.
- myftpserver is expected to be able to serve another data transfer session without restart.
- How can the myftpclient tell myftpserver that the current data transfer session is finished? EndPacket.

Implementation Hints:Terminate



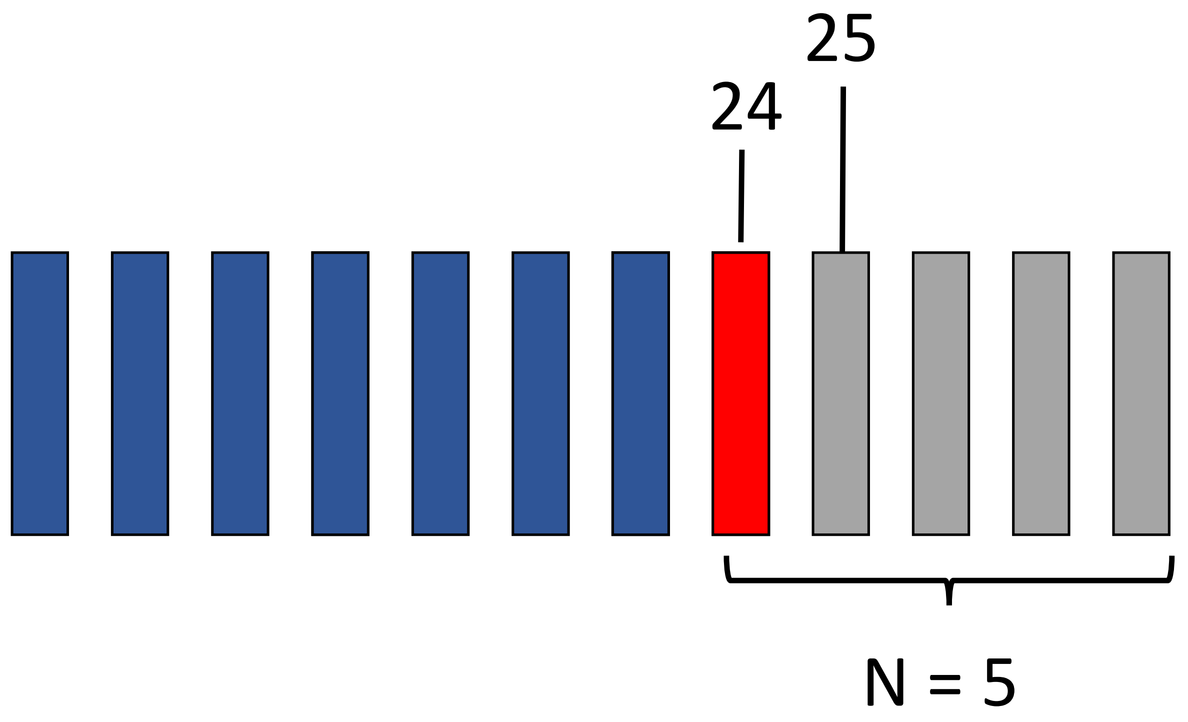
- When to send EndPacket?
  - When there is no un-acked DataPackets in current window!
- Suppose
  - base = 23
  - nextseqnum = 24
  - N = 5

Implementation Hints:Terminate



- Suppose
  - base = 23
  - nextseqnum = 24
  - N = 5
- Suppose 23 is the last data packet, even if there is no other application request to send data, we cannot send EndPacket.
- Because we haven’t receive the acknowledge that the last DataPacket reaches the server

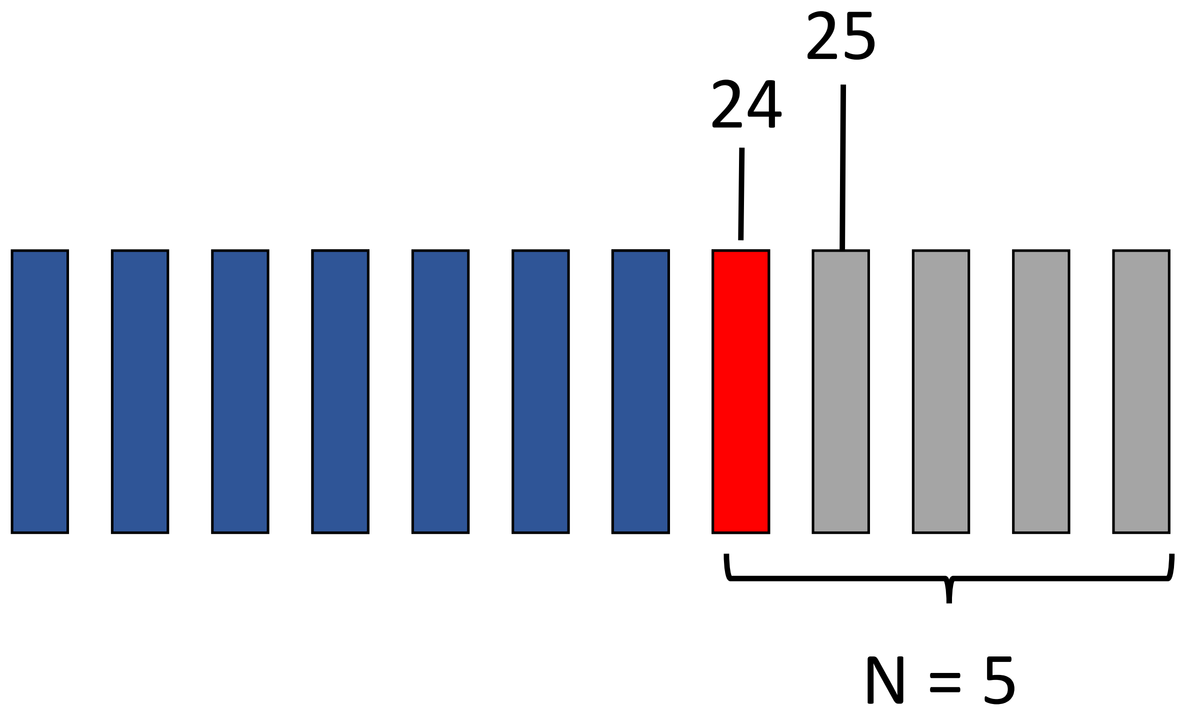
Implementation Hints:Terminate



- When sender receives the ACK(23), then the sender can send EndPacket with seqNum = 24.
  - base = 24
  - nextseqnum = 25

🟥 Sent, un-ACKed EndPacket

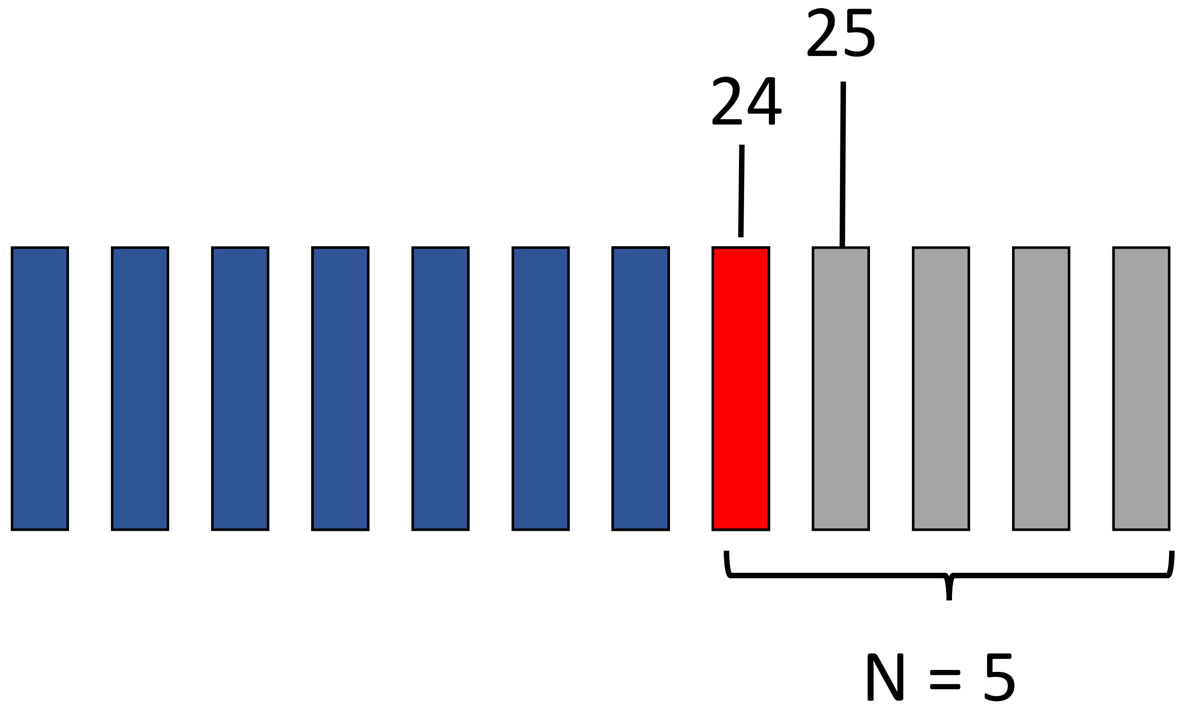
Implementation Hints:Terminate



- Now
  - base = 24
  - nextseqnum = 25
  - N = 5
- If sender receives ACK(24), then client can terminate gracefully

🟥 Sent, un-ACKed EndPacket

Implementation Hints:Terminate



🟥 Sent, un-ACKed EndPacket

- Now
  - base = 24
  - nextseqnum = 25
  - N = 5
- If sender does not receive ACK(24) after 3 times re-transmission, client terminates and report an error.

Implementation Hints: Terminate

- How does receiver respond to the EndPacket?
- Receiver reset everything it needs to prepare for another data transfer session.

Implementation Hints: multi-threading

- Sender side: at least 2 threads
  - a thread to receive AckPackets
  - a thread for timer (you may use as many threads as you need for timer)
- Optional threads
  - a thread to send DataPackets
- This is an implementation issue and you are free to design your own threads

Implementation Hints: multi-threading

- Receiver side: at least 1 threads
  - a thread to receive DataPacket and EndPacket

Other Issues

- Assumptions:
  - We only consider upload operation from client to the server
  - Each time there is only one client which uploads file to the server
- Command line format:
  - ./myftpserver <port number>
  - ./myftpclient <server ip addr> <server port> <file> <N> <timeout>

Other Issues

- Please do not modify the application framework.
- You are free to design your Selective Repeat layer.
- Please use the provided functions to control the action of your Selective Repeat layer.
  - mygbn_init_sender()
  - mygbn_send()
  - mygbn_close_sender()
  - mygbn_init_receiver()
  - mygbn_recv()
  - mygbn_close_receiver()

Other Issues

- Submission
  - Makefile
  - All the source code
- TA will check whether you modify the original myftp code.
- Please make sure your Makefile works.
- Please make sure your Makefile create
  - myftpserver
  - myftpclient
- c/c++ implementations are allowed.

Other Issues

- If you still have question, please post your question in our piazza group