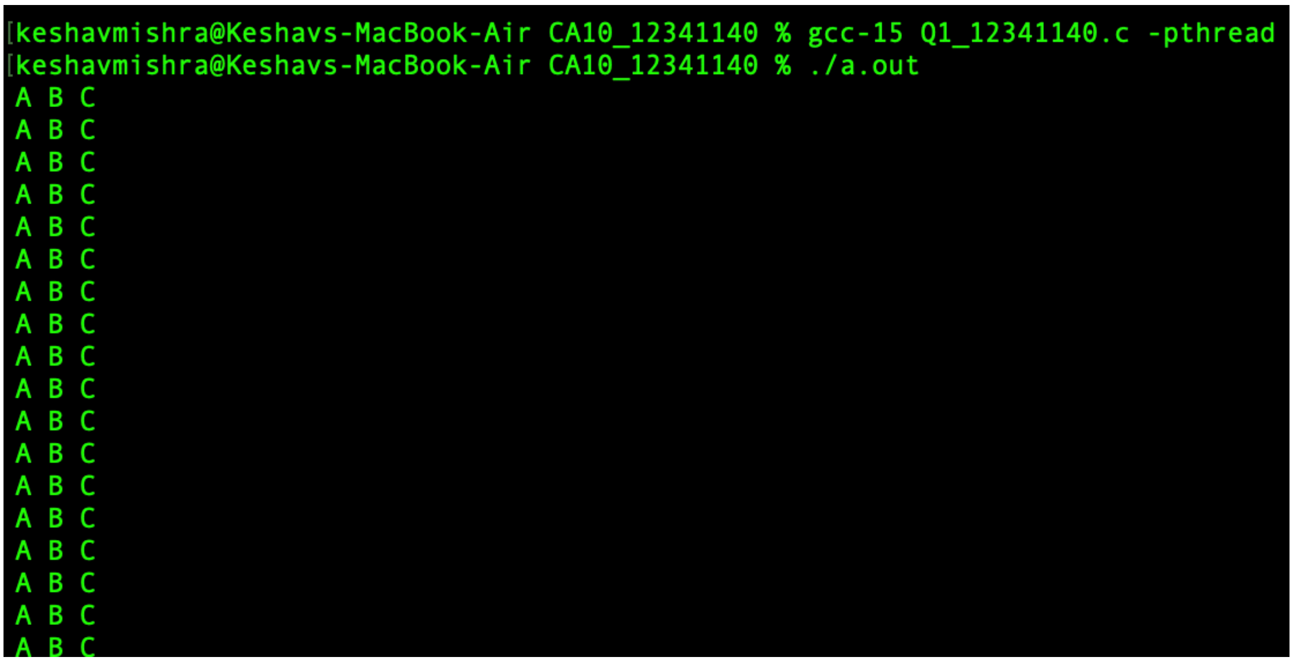
Q1

Code:
```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#define N 102  // number of cycles

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

int turn = 0; // 0 -> A, 1 -> B, 2 -> C

void *printA(void *arg) {
    for (int i = 0; i < N; i++) {
        pthread_mutex_lock(&lock);
        while (turn != 0)
            pthread_cond_wait(&cond, &lock);

        printf("A ");
        fflush(stdout);

        turn = 1;
        pthread_cond_broadcast(&cond);
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

void *printB(void *arg) {
    for (int i = 0; i < N; i++) {
        pthread_mutex_lock(&lock);
        while (turn != 1)
            pthread_cond_wait(&cond, &lock);

        printf("B ");
        fflush(stdout);

        turn = 2;
        pthread_cond_broadcast(&cond);
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

void *printC(void *arg) {
    for (int i = 0; i < N; i++) {
        pthread_mutex_lock(&lock);
        while (turn != 2)
            pthread_cond_wait(&cond, &lock);
        printf("C\n");
        fflush(stdout);

        turn = 0;
        pthread_cond_broadcast(&cond);
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

int main() {
    pthread_t tA, tB, tC;
    pthread_create(&tA, NULL, printA, NULL);
    pthread_create(&tB, NULL, printB, NULL);
    pthread_create(&tC, NULL, printC, NULL);

    pthread_join(tA, NULL);
    pthread_join(tB, NULL);
    pthread_join(tC, NULL);
    return 0;
}
```

Screenshot of output:



```
[keshavmishra@Keshavs-MacBook-Air CA10_12341140 % gcc-15 Q1_12341140.c -pthread
[keshavmishra@Keshavs-MacBook-Air CA10_12341140 % ./a.out
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
 A B C
```

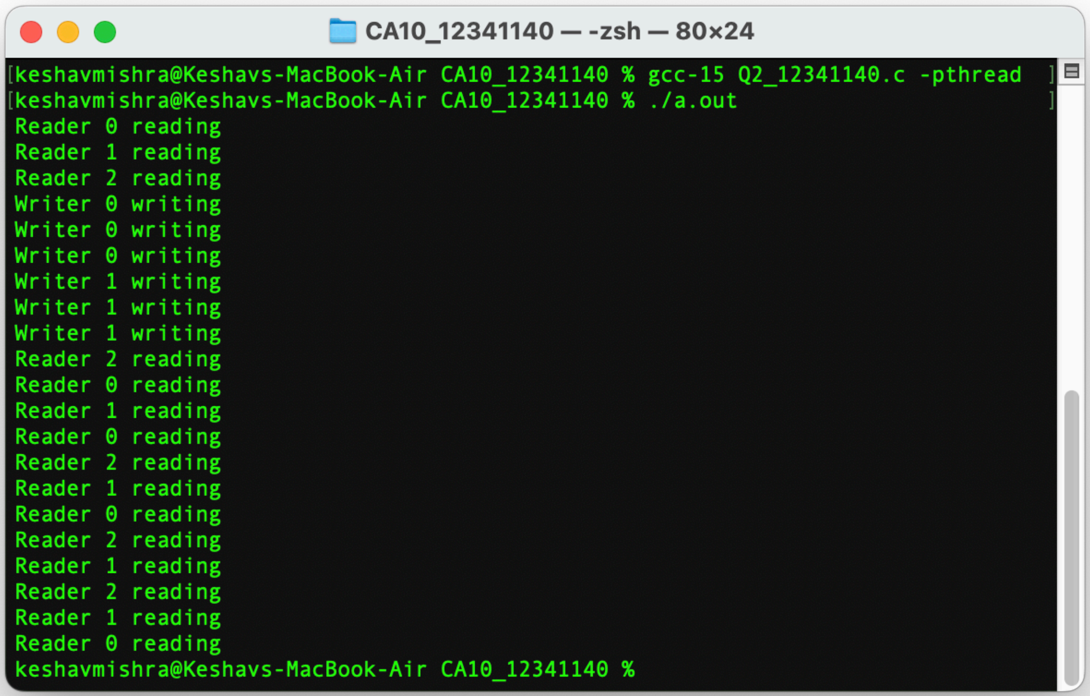
Explanation:
Readers wait if a writer is active or waiting, giving writers priority to avoid starvation.
Writers gain exclusive access to the shared resource for safe updates.
After writing, preference is again given to waiting writers before allowing new readers.

Q2

Code:
```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t can_read = PTHREAD_COND_INITIALIZER;
pthread_cond_t can_write = PTHREAD_COND_INITIALIZER;

int read_count = 0;
int write_count = 0;
int waiting_writers = 0;

void start_read() {
    pthread_mutex_lock(&lock);

    // Readers must wait if a writer is writing OR writers are waiting
    while (write_count > 0 || waiting_writers > 0) {
        pthread_cond_wait(&can_read, &lock);
    }

    read_count++;
    pthread_mutex_unlock(&lock);
}

void end_read() {
    pthread_mutex_lock(&lock);
    read_count--;

    // If no readers left, give chance to waiting writers
    if (read_count == 0)
        pthread_cond_signal(&can_write);

    pthread_mutex_unlock(&lock);
}

void start_write() {
    pthread_mutex_lock(&lock);
    waiting_writers++;

    // Writer must wait if another writer is active OR readers are reading
    while (write_count > 0 || read_count > 0) {
        pthread_cond_wait(&can_write, &lock);
    }

    waiting_writers--;
    write_count = 1;
    pthread_mutex_unlock(&lock);
}
```

```
void end_write() {
    pthread_mutex_lock(&lock);
    write_count = 0;

    // Writers get priority if waiting, else wake all readers
    if (waiting_writers > 0)
        pthread_cond_signal(&can_write);
    else
        pthread_cond_broadcast(&can_read);

    pthread_mutex_unlock(&lock);
}

void *reader(void *id) {
    for (int i = 0; i < 5; i++) {
        start_read();
        printf("Reader %ld reading\n", (long)id);
        usleep(100000);
        end_read();
    }
    return NULL;
}

void *writer(void *id) {
    for (int i = 0; i < 3; i++) {
        start_write();
        printf("Writer %ld writing\n", (long)id);
        usleep(150000);
        end_write();
    }
    return NULL;
}

int main() {
    pthread_t r[3], w[2];

    for (long i = 0; i < 3; i++)
        pthread_create(&r[i], NULL, reader, (void *)i);

    for (long i = 0; i < 2; i++)
        pthread_create(&w[i], NULL, writer, (void *)i);

    for (int i = 0; i < 3; i++)
        pthread_join(r[i], NULL);

    for (int i = 0; i < 2; i++)
        pthread_join(w[i], NULL);

    return 0;
}
```

Screenshot of output:



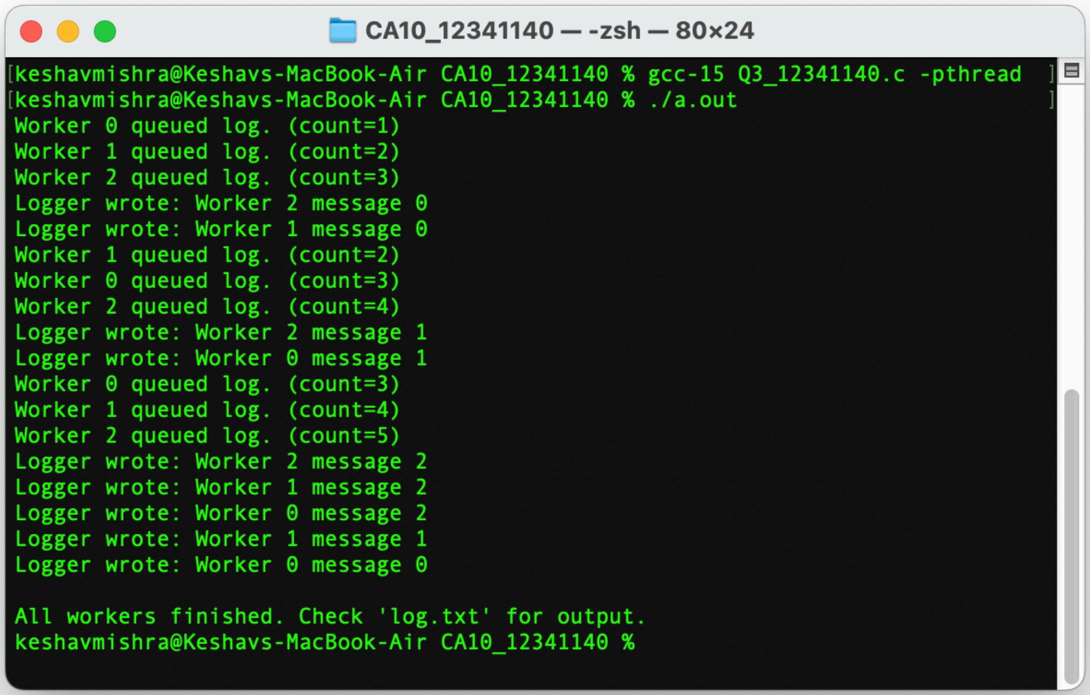
```
[keshavmishra@Keshavs-MacBook-Air CA10_12341140 % gcc-15 Q2_12341140.c -pthread  ]
[keshavmishra@Keshavs-MacBook-Air CA10_12341140 % ./a.out                         ]
Reader 0 reading
Reader 1 reading
Reader 2 reading
Writer 0 writing
Writer 0 writing
Writer 0 writing
Writer 1 writing
Writer 1 writing
Writer 1 writing
Reader 2 reading
Reader 0 reading
Reader 1 reading
Reader 0 reading
Reader 2 reading
Reader 1 reading
Reader 0 reading
Reader 2 reading
Reader 1 reading
Reader 2 reading
Reader 1 reading
Reader 0 reading
keshavmishra@Keshavs-MacBook-Air CA10_12341140 %
```

Explanation:
Readers wait if a writer is active or waiting, giving writers priority to avoid starvation.
Writers gain exclusive access to the shared resource for safe updates.
After writing, preference is again given to waiting writers before allowing new readers.

Q3

Code:
```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define MAX_LOGS 10

char *log_buffer[MAX_LOGS];
int count = 0;

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t has_logs = PTHREAD_COND_INITIALIZER;
pthread_cond_t has_space = PTHREAD_COND_INITIALIZER;

void *worker(void *id) {
    for (int i = 0; i < 3; i++) {
        char msg[64];
        sprintf(msg, "Worker %ld message %d", (long)id, i);

        pthread_mutex_lock(&lock);
        while (count == MAX_LOGS)
            pthread_cond_wait(&has_space, &lock);

        log_buffer[count++] = strdup(msg);
        printf("Worker %ld queued log. (count=%d)\n", (long)id, count);

        pthread_cond_signal(&has_logs);
        pthread_mutex_unlock(&lock);

        usleep(100000);
    }
    return NULL;
}

void *logger(void *arg) {
    FILE *f = fopen("log.txt", "w");
    if (!f) {
        perror("fopen");
        return NULL;
    }

    while (1) {
        pthread_mutex_lock(&lock);
        while (count == 0)
            pthread_cond_wait(&has_logs, &lock);
        char *msg = log_buffer[--count];
        pthread_cond_signal(&has_space);
        pthread_mutex_unlock(&lock);

        fprintf(f, "%s\n", msg);
        fflush(f);
        printf("Logger wrote: %s\n", msg);
        free(msg);
        usleep(50000);
    }

    fclose(f);
    return NULL;
}

int main() {
    pthread_t log_thread, workers[3];
    pthread_create(&log_thread, NULL, logger, NULL);
    for (long i = 0; i < 3; i++)
        pthread_create(&workers[i], NULL, worker, (void *)i);

    for (int i = 0; i < 3; i++)
        pthread_join(workers[i], NULL);

    sleep(1);
    pthread_cancel(log_thread);
    pthread_join(log_thread, NULL);

    printf("\nAll workers finished. Check 'log.txt' for output.\n");
    return 0;
}
```

Screenshot of output:


CA10_12341140 — -zsh — 80×24

```
keshavmishra@Keshavs-MacBook-Air CA10_12341140 % gcc-15 Q3_12341140.c -pthread
keshavmishra@Keshavs-MacBook-Air CA10_12341140 % ./a.out
Worker 0 queued log. (count=1)
Worker 1 queued log. (count=2)
Worker 2 queued log. (count=3)
Logger wrote: Worker 2 message 0
Logger wrote: Worker 1 message 0
Worker 1 queued log. (count=2)
Worker 0 queued log. (count=3)
Worker 2 queued log. (count=4)
Logger wrote: Worker 2 message 1
Logger wrote: Worker 0 message 1
Worker 0 queued log. (count=3)
Worker 1 queued log. (count=4)
Worker 2 queued log. (count=5)
Logger wrote: Worker 2 message 2
Logger wrote: Worker 1 message 2
Logger wrote: Worker 0 message 2
Logger wrote: Worker 1 message 1
Logger wrote: Worker 0 message 0

All workers finished. Check 'log.txt' for output.
keshavmishra@Keshavs-MacBook-Air CA10_12341140 %
```

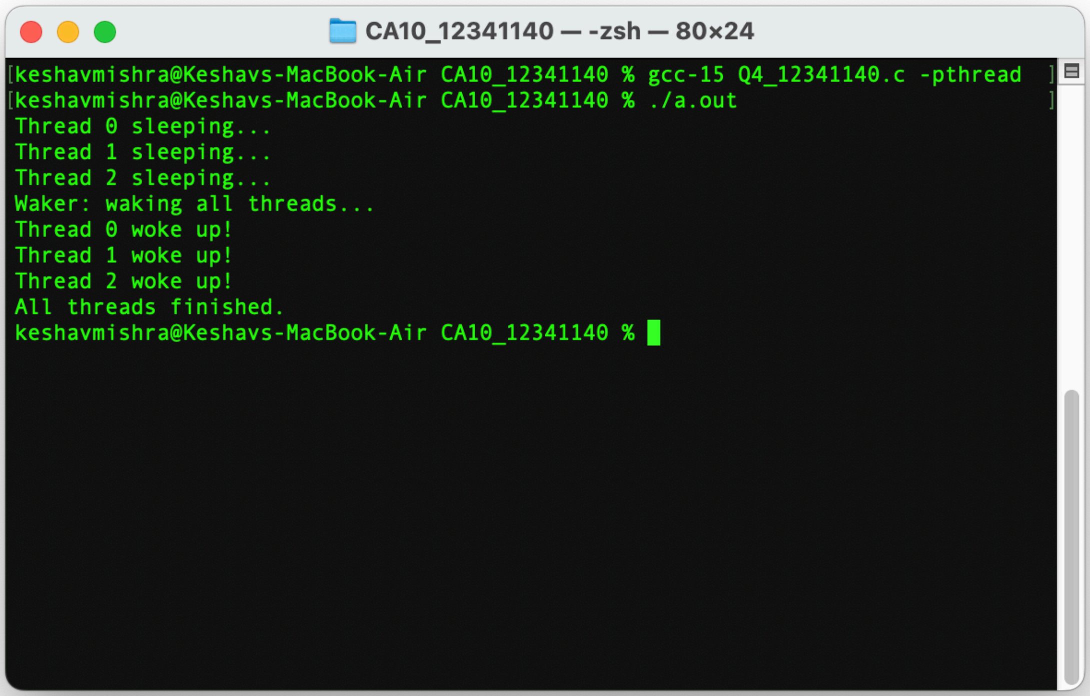
Explanation:
Mutex and condition variables synchronize workers and the logger for proper coordination.
The logger waits when there are no logs to write, and workers wait when the buffer is full.
This prevents busy-waiting, ensures all messages are logged, and avoids message loss.

Q4

Code:
```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#define NTHREADS 3

struct thread_event {
   pthread_mutex_t m;
   pthread_cond_t c;
   int awake;
};

struct thread_event events[NTHREADS];

void *sleeper(void *id) {
   long tid = (long)id;
   pthread_mutex_lock(&events[tid].m);

   while (!events[tid].awake) {
      printf("Thread %ld sleeping...\n", tid);
      pthread_cond_wait(&events[tid].c, &events[tid].m);
   }

   printf("Thread %ld woke up!\n", tid);

   pthread_mutex_unlock(&events[tid].m);
   return NULL;
}

void *waker(void *arg) {
   sleep(2);

   printf("Waker: waking all threads...\n");

   for (int i = 0; i < NTHREADS; i++) {
      pthread_mutex_lock(&events[i].m);
      events[i].awake = 1;
      pthread_cond_signal(&events[i].c);
      pthread_mutex_unlock(&events[i].m);
   }
```

```
    return NULL;
}

int main() {
    pthread_t t[NTHREADS], w;

    for (int i = 0; i < NTHREADS; i++) {
        events[i].awake = 0;
        pthread_mutex_init(&events[i].m, NULL);
        pthread_cond_init(&events[i].c, NULL);
        pthread_create(&t[i], NULL, sleeper, (void *)(long)i);
    }

    pthread_create(&w, NULL, waker, NULL);
    pthread_join(w, NULL);

    for (int i = 0; i < NTHREADS; i++)
        pthread_join(t[i], NULL);

    printf("All threads finished.\n");
    return 0;
}
```

Screenshot of output:



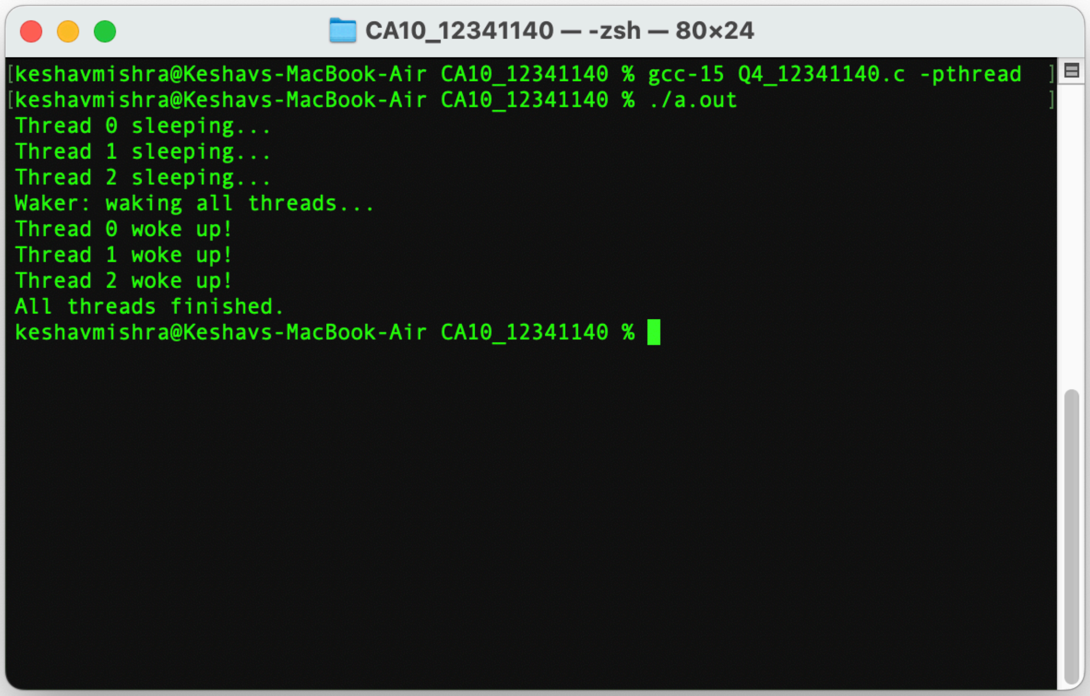
```
[keshavmishra@Keshavs-MacBook-Air CA10_12341140 % gcc-15 Q4_12341140.c -pthread ]
[keshavmishra@Keshavs-MacBook-Air CA10_12341140 % ./a.out                        ]
 Thread 0 sleeping...
 Thread 1 sleeping...
 Thread 2 sleeping...
 Waker: waking all threads...
 Thread 0 woke up!
 Thread 1 woke up!
 Thread 2 woke up!
 All threads finished.
 keshavmishra@Keshavs-MacBook-Air CA10_12341140 %
```

Explanation:
Each sleeping thread waits on its condition variable until the waker sets its awake flag and signals it.
This ensures threads actually sleep instead of busy-waiting, saving CPU time.
The waker properly synchronizes all threads, allowing them to wake up and proceed safely.

Q5

Code:
```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#define MAX_JOBS 5
#define NUM_WORKERS 3

int jobs[MAX_JOBS];
int count = 0;

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t has_jobs = PTHREAD_COND_INITIALIZER;
pthread_cond_t has_space = PTHREAD_COND_INITIALIZER;

void *dispatcher(void *arg) {
    int job_id = 1;
    while (job_id <= 10) {
        pthread_mutex_lock(&lock);

        while (count == MAX_JOBS)
            pthread_cond_wait(&has_space, &lock);

        jobs[count++] = job_id;
        printf("Dispatcher added job %d (count=%d)\n", job_id, count);

        pthread_cond_signal(&has_jobs);
        pthread_mutex_unlock(&lock);

        job_id++;
        usleep(100000);
    }
    return NULL;
}

void *worker(void *arg) {
    long id = (long)arg;

    while (1) {
        pthread_mutex_lock(&lock);

        while (count == 0)
            pthread_cond_wait(&has_jobs, &lock);

        int job = jobs[--count];
        printf("Worker %ld processing job %d (remaining=%d)\n", id, job, count);

        pthread_cond_signal(&has_space);
        pthread_mutex_unlock(&lock);
        usleep(200000);
    }
    return NULL;
}

int main() {
    pthread_t disp, workers[NUM_WORKERS];

    pthread_create(&disp, NULL, dispatcher, NULL);
    for (long i = 0; i < NUM_WORKERS; i++)
        pthread_create(&workers[i], NULL, worker, (void *)i);

    pthread_join(disp, NULL);
    sleep(2);

    printf("All jobs dispatched. Exiting...\n");
    return 0;
}
```

Screenshot of output:



```
CA10_12341140 — -zsh — 80×24
[keshavmishra@Keshavs-MacBook-Air CA10_12341140 % gcc-15 Q4_12341140.c -pthread  ]
[keshavmishra@Keshavs-MacBook-Air CA10_12341140 % ./a.out                         ]
Thread 0 sleeping...
Thread 1 sleeping...
Thread 2 sleeping...
Waker: waking all threads...
Thread 0 woke up!
Thread 1 woke up!
Thread 2 woke up!
All threads finished.
keshavmishra@Keshavs-MacBook-Air CA10_12341140 %
```

Explanation:
Mutexes and condition variables eliminate busy waiting by making the dispatcher wait when the job buffer is full and workers wait when it's empty.
pthread_cond_signal and pthread_cond_wait ensure proper coordination between job producers and consumers.
This prevents CPU wastage and ensures smooth, synchronized job scheduling between dispatcher and workers.