CS 140 Project 2 Documentation

Aaron Jude Tanael
202004275

January 2023

Contents

Introduction

The project documentation contains the implementation details behind the parallelized `grep` runner written in C for Linux.

The documentation proper contains the following:

1. All references used with purpose specified.
2. Walkthrough of code execution with sample run having at least `N = 2` on a multicore machine, one `PRESENT`, five `ABSENT`s, and six `DIR`s for `multithreaded.c`.
3. Explanation of how the task queue was implemented using the synchronization construct of choice, including how race conditions were handled.
4. Explanation of how each worker knows when to terminate (i.e., mechanism that determines and synchronizes that no more content will be enqueued), including how race conditions were handled.

Note that I only reached the `multithreaded` part of this project.

For the actual working C files for the implementation, please refer to the Github Classroom repository through this link: `https://github.com/UPD-CS140/cs140221project2-a-tanael`. For the documentation video, please refer to this Google Drive link: `https://drive.google.com/file/d/1F9OOYUi-McqaRGoenh1SmFnh32Y8n6m5/view?usp=share_link`

The project documentation is made using LaTeX.

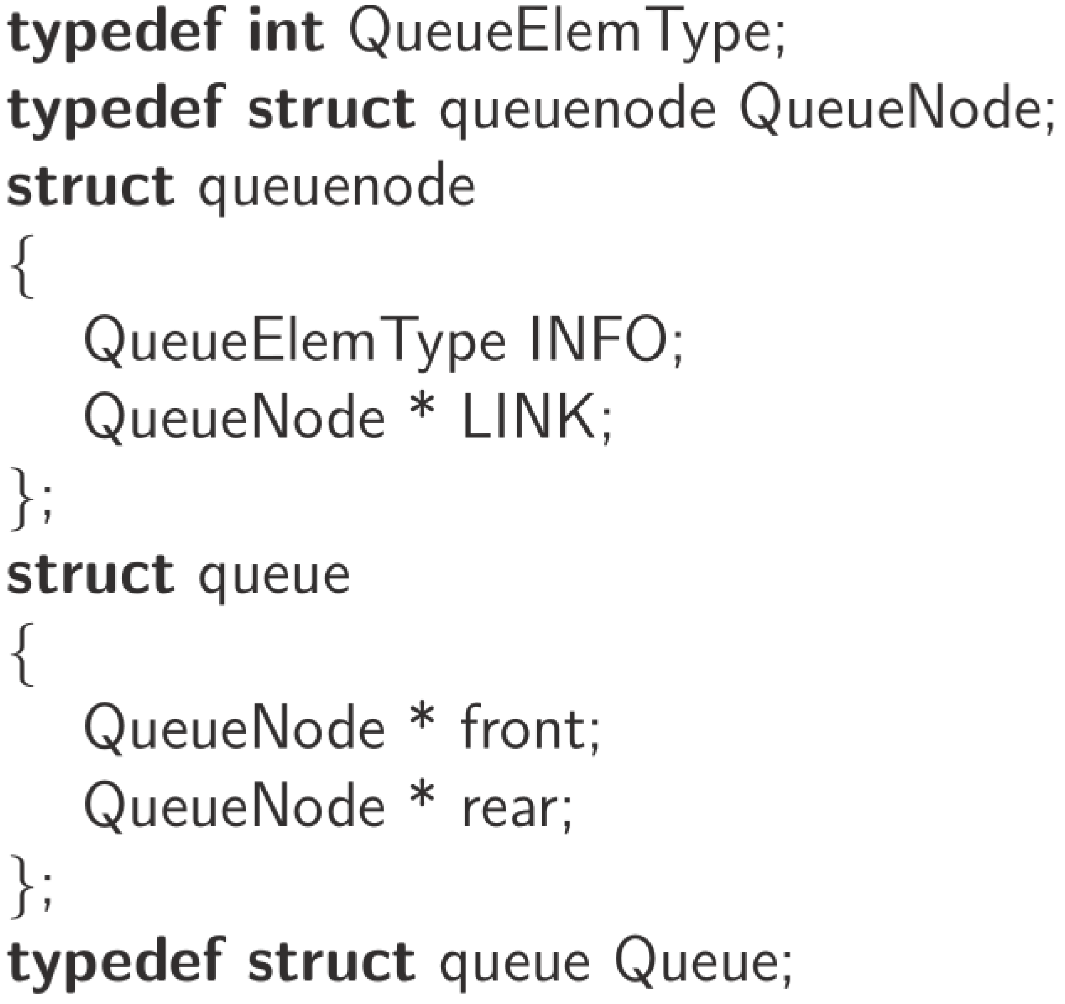
Documentation proper

1. All references used with purpose specified (*if any; otherwise, explicitly state that you did not use any external resource*).

   I used some major references for implementing the task queue and synchronization primitives to implement the parallelized `grep` runner:

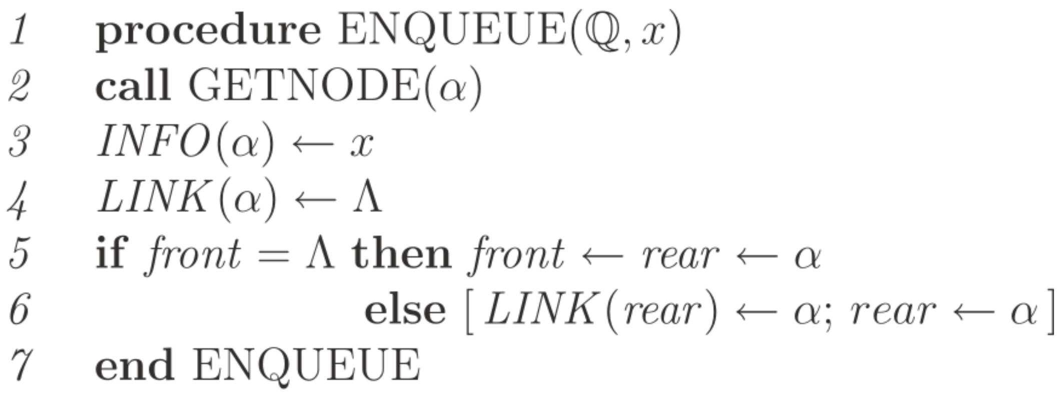
   - I used Quiwa's[3] book on *Data Structures* for implementing the structure and methods for the task queue. More specifically, I used the pseudocodes of those methods as primary reference on how I would structure my implementation of the task queue.

     The pseudocodes are as follows:



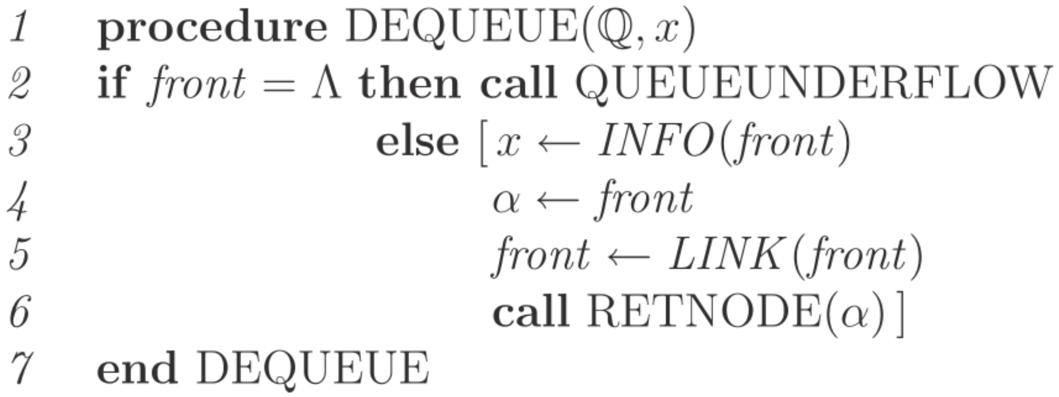
```
typedef int QueueElemType;
typedef struct queuenode QueueNode;
struct queuenode
{
    QueueElemType INFO;
    QueueNode * LINK;
};
struct queue
{
    QueueNode * front;
    QueueNode * rear;
};
typedef struct queue Queue;
```

Figure 2.1: Linked list implementation of a queue



```
procedure ENQUEUE($\mathbb{Q}, x$)
call GETNODE($\alpha$)
$INFO(\alpha) \leftarrow x$
$LINK(\alpha) \leftarrow \Lambda$
if $front = \Lambda$ then $front \leftarrow rear \leftarrow \alpha$
                    else [ $LINK(rear) \leftarrow \alpha$; $rear \leftarrow \alpha$ ]
end ENQUEUE
```

Figure 2.2: Enqueueing into a linked list queue

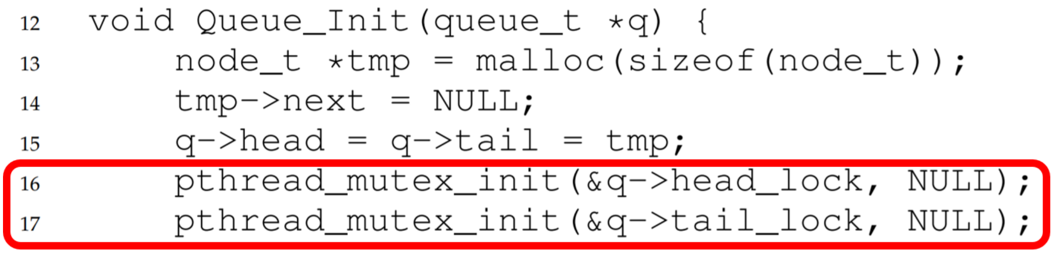


```
procedure DEQUEUE($\mathbb{Q}, x$)
if $front = \Lambda$ then call QUEUEUNDERFLOW
                    else [ $x \leftarrow INFO(front)$
                           $\alpha \leftarrow front$
                           $front \leftarrow LINK(front)$
                           call RETNODE($\alpha$) ]
end DEQUEUE
```

Figure 2.3: Dequeueing from a linked list queue. Also includes a subroutine checking if the linked list queue is empty.

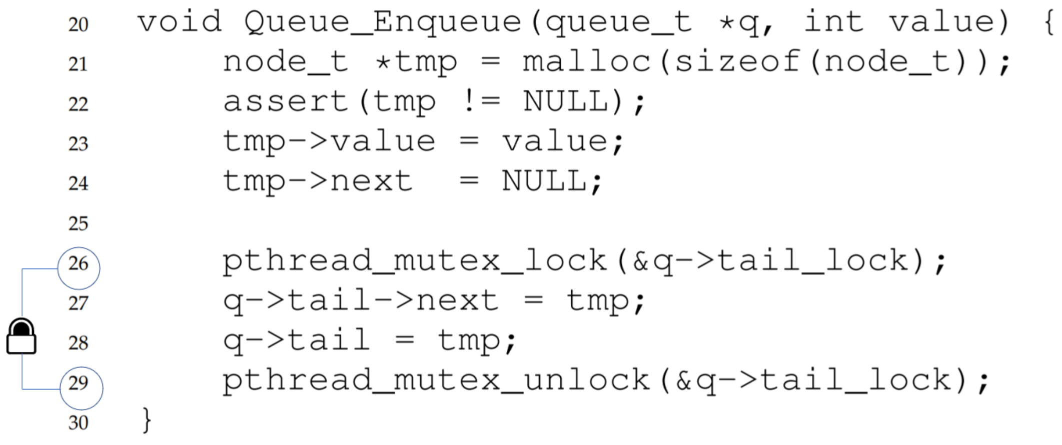
- I used the CS 140 Team 22.1 Lecture 16 regarding **lock-based concurrent data structures** and the CS 140 Team 22.1 Lecture 18 regarding **semaphores** for handling the task queue between different threads for parallelism. I also used the code blocks provided by OSTEP[1] in the said lectures for straightforward implementations, although I added necessary modifications to fit the specifications of the project.

  The code blocks are as follows:



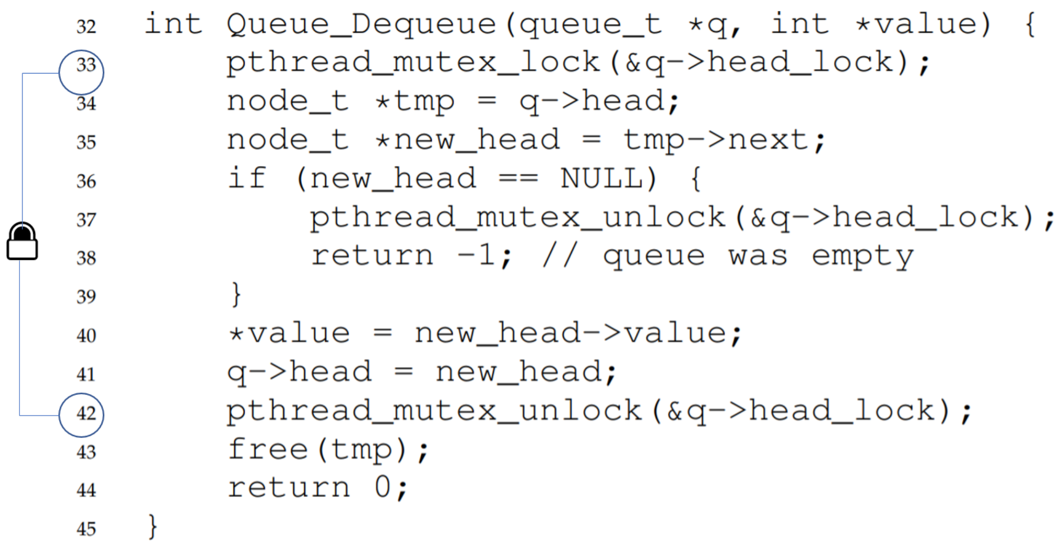
```
void Queue_Init(queue_t *q) {
    node_t *tmp = malloc(sizeof(node_t));
    tmp->next = NULL;
    q->head = q->tail = tmp;
    pthread_mutex_init(&q->head_lock, NULL);
    pthread_mutex_init(&q->tail_lock, NULL);
```

Figure 2.4: Initializing the concurrent task queue



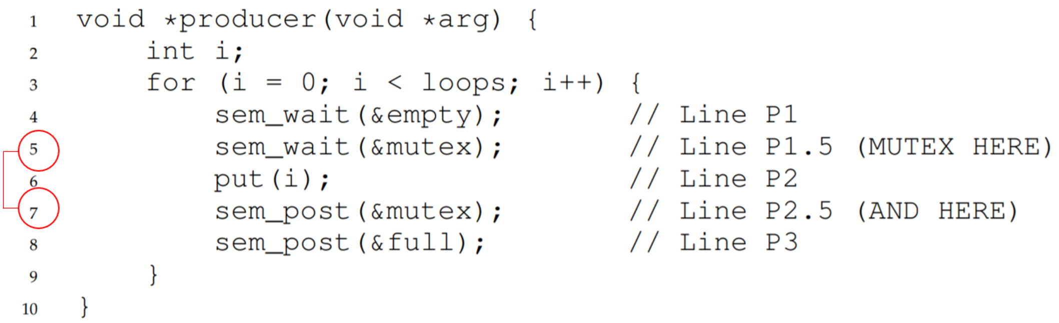
```
void Queue_Enqueue(queue_t *q, int value) {
    node_t *tmp = malloc(sizeof(node_t));
    assert(tmp != NULL);
    tmp->value = value;
    tmp->next  = NULL;

    pthread_mutex_lock(&q->tail_lock);
    q->tail->next = tmp;
    q->tail = tmp;
    pthread_mutex_unlock(&q->tail_lock);
}
```

Figure 2.5: `ENQUEUE` method for the concurrent task queue



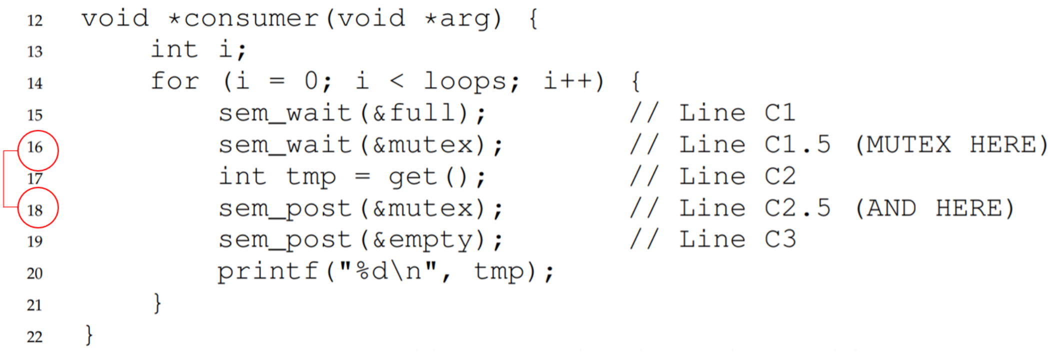
```
int Queue_Dequeue(queue_t *q, int *value) {
    pthread_mutex_lock(&q->head_lock);
    node_t *tmp = q->head;
    node_t *new_head = tmp->next;
    if (new_head == NULL) {
        pthread_mutex_unlock(&q->head_lock);
        return -1; // queue was empty
    }
    *value = new_head->value;
    q->head = new_head;
    pthread_mutex_unlock(&q->head_lock);
    free(tmp);
    return 0;
}
```

Figure 2.6: `DEQUEUE` method for the concurrent task queue



```
void *producer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        sem_wait(&empty);        // Line P1
        sem_wait(&mutex);        // Line P1.5 (MUTEX HERE)
        put(i);                  // Line P2
        sem_post(&mutex);        // Line P2.5 (AND HERE)
        sem_post(&full);         // Line P3
    }
}
```

Figure 2.7: Characteristics of the `producer` thread for the bounded buffer problem



```
void *consumer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        sem_wait(&full);        // Line C1
        sem_wait(&mutex);       // Line C1.5 (MUTEX HERE)
        int tmp = get();        // Line C2
        sem_post(&mutex);       // Line C2.5 (AND HERE)
        sem_post(&empty);       // Line C3
        printf("%d\n", tmp);
    }
}
```

Figure 2.8: Characteristics of the `consumer` thread for the bounded buffer problem

- I used the CS 140 Team 22.1 Lecture 22 regarding **APIs and system calls related to files and directories** for traversing the directory level of `rootpath` (`.` in the lecture), as included in the project specifications. Again, I included the code block for traversing the directory tree provided by OSTEP[1] in the said lectures for straightforward implementations with modifications.

  The code block is as follows:

```
int main(int argc, char *argv[]) {
    DIR *dp = opendir(".");
    assert(dp != NULL);
    struct dirent *d;
    while ((d = readdir(dp)) != NULL) {
        printf("%lu %s\n", (unsigned long) d->d_ino,
               d->d_name);
    }
    closedir(dp);
    return 0;
}
```

Figure 2.9: Traversing the directory level from `.` as `rootpath`

  I will discuss later how this code block helped me formulate the code for traversing the entire directory tree starting from `rootpath`.

- I used the Linux man pages[2] to gain information about **APIs and system calls related to files and directories**, as well as necessary Linux commands (e.g., `snprintf`), so that I can easily implement the automation of `grep` execution while traversing the directories.

  The commands are as follows:

  - `strcmp`
  - `printf`
  - `pthread_create`
  - `pthread_join`
  - `opendir`
  - `readdir`
  - `closedir`
  - `getcwd`
  - `system`
  - `malloc`
  - `free`
  - `sem_init`
  - `sem_wait`
  - `sem_post`
  - `sem_destroy`
  - `exit`

For the bibliography-style enumeration of the references used, please refer to the last page of this document.

2. For each version submitted (excluding single-process, single-threaded version):

   (a) Walkthrough of code execution with sample run having at least `N = 2` on a multicore machine, one `PRESENT`, five `ABSENT`s, and six `DIR`.

   Before we begin, we try to set up our necessary directories and `.txt` files for our test case, following this directory tree:

```
rootpath/
  initpath/
    path 1/
      path 4/
        path 7/
          hello.txt <-- hello is located here!
        hello.txt
      hello.txt
    path 2/
      path 5/
        hello.txt
      hello.txt
    path 3/
      path 6/
        hello.txt
      hello.txt
    hello.txt
  hello.txt
single.c
multithreaded.c
```

Figure 2.10: Directory tree for the sample run

The specs of the machine that I will use for the demonstration has `N = 4` cores. We expect that upon running `./multithreaded 4 rootpath hello`, one `PRESENT`, eight `ABSENT`s, and nine `DIR`s must be present in the standard output to be shown in the console, with threads 0 to 3 having a well-distributed amount of work. For different run instances of the program, we should expect a bit of randomness in the output, since threads have to be scheduled differently to work on tasks in the task queue.

In this case I am about to show the walkthrough of the `multithreaded.c` code using the sample run. Note that I used the shared folder as the current working directory for this sample run, since I was using Oracle VM Virtualbox to accomplish this project.

Initial state

```
[0] DIR /media/sf_cs140project2/rootpath
[0] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[2] DIR /media/sf_cs140project2/rootpath/initpath
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.11: Sample output for the sample run

rootpath/				

Figure 2.12: Task queue for the sample run, prior to the code execution walkthrough

```
rootpath/
  initpath/
    path 1/
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/
      path 5/
        hello.txt
      hello.txt
    path 3/
      path 6/
        hello.txt
      hello.txt
    hello.txt
  hello.txt
single.c
multithreaded.c
```

Figure 2.13: State of the directory tree prior to worker assignment

DIR .../rootpath

```
[X] DIR /media/sf_cs140project2/rootpath
[0] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[2] DIR /media/sf_cs140project2/rootpath/initpath
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.14: Current code execution

Figure 2.15: Current state of the task queue

```
rootpath/ <-- Worker 0
  initpath/
    path 1/
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/
      path 5/
        hello.txt
      hello.txt
    path 3/
      path 6/
        hello.txt
      hello.txt
    hello.txt
  hello.txt
single.c
multithreaded.c
```

Figure 2.16: Current state of the workers in the directory tree

ABSENT .../rootpath/hello.txt

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[2] DIR /media/sf_cs140project2/rootpath/initpath
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.17: Current code execution

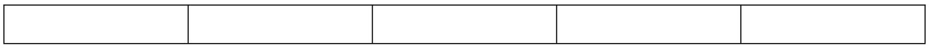


Figure 2.18: Current state of the task queue

```
rootpath/ <-- X
  initpath/
    path 1/
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/
      path 5/
        hello.txt
      hello.txt
    path 3/
      path 6/
        hello.txt
      hello.txt
    hello.txt
  hello.txt <-- Worker 0
single.c
multithreaded.c
```

Figure 2.19: Current state of the workers in the directory tree

ENQUEUE .../rootpath/initpath

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[2] DIR /media/sf_cs140project2/rootpath/initpath
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.20: Current code execution

initpath/				

Figure 2.21: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- Worker 0
    path 1/
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/
      path 5/
        hello.txt
      hello.txt
    path 3/
      path 6/
        hello.txt
      hello.txt
    hello.txt
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.22: Current state of the workers in the directory tree

DIR .../rootpath/initpath

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.23: Current code execution

Figure 2.24: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- Worker 2
    path 1/
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/
      path 5/
        hello.txt
      hello.txt
    path 3/
      path 6/
        hello.txt
      hello.txt
    hello.txt
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.25: Current state of the workers in the directory tree

ABSENT .../rootpath/initpath/hello.txt

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.26: Current code execution

Figure 2.27: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/
      path 5/
        hello.txt
      hello.txt
    path 3/
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- Worker 2
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.28: Current state of the workers in the directory tree

ENQUEUE .../rootpath/initpath/path 1

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.29: Current code execution

path 1				

Figure 2.30: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- Worker 2
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/
      path 5/
        hello.txt
      hello.txt
    path 3/
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.31: Current state of the workers in the directory tree

ENQUEUE .../rootpath/initpath/path 2

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.32: Current code execution

path 1	path 2			

Figure 2.33: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- ...
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/ <-- Worker 2
      path 5/
        hello.txt
      hello.txt
    path 3/
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.34: Current state of the workers in the directory tree

ENQUEUE .../rootpath/initpath/path 3

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.35: Current code execution

path 1	path 2	path 3		

Figure 2.36: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- ...
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/ <-- ...
      path 5/
        hello.txt
      hello.txt
    path 3/ <-- Worker 2
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.37: Current state of the workers in the directory tree

DIR .../rootpath/initpath/path 1

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.38: Current code execution

path 2	path 3			

Figure 2.39: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- Worker 1
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt
    path 2/ <-- ...
      path 5/
        hello.txt
      hello.txt
    path 3/ <-- ...
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.40: Current state of the workers in the directory tree

ABSENT .../rootpath/initpath/path 1/hello.txt

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[1] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.41: Current code execution

path 2	path 3			

Figure 2.42: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- Worker 1
    path 2/ <-- ...
      path 5/
        hello.txt
      hello.txt
    path 3/ <-- ...
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.43: Current state of the workers in the directory tree

ENQUEUE .../rootpath/initpath/path 1/path 4

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.44: Current code execution

path 2	path 3	path 4		

Figure 2.45: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- Worker 1
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- ...
      path 5/
        hello.txt
      hello.txt
    path 3/ <-- ...
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.46: Current state of the workers in the directory tree

DIR .../rootpath/initpath/path 2

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.47: Current code execution

path 3	path 4			

Figure 2.48: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- ...
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- Worker 3
      path 5/
        hello.txt
      hello.txt
    path 3/ <-- ...
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.49: Current state of the workers in the directory tree

DIR .../rootpath/initpath/path 3

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[2] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.50: Current code execution

path 4				

Figure 2.51: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- ...
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- Worker 3
      path 5/
        hello.txt
      hello.txt
    path 3/ <-- Worker 0
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.52: Current state of the workers in the directory tree

DIR .../rootpath/initpath/path 1/path 4

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.53: Current code execution

Figure 2.54: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- Worker 2
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- Worker 3
      path 5/
        hello.txt
      hello.txt
    path 3/ <-- Worker 0
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.55: Current state of the workers in the directory tree

ABSENT .../rootpath/initpath/path 2/hello.txt

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[3] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.56: Current code execution

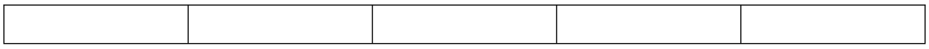


Figure 2.57: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- Worker 2
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- X
      path 5/
        hello.txt
      hello.txt <-- Worker 3
    path 3/ <-- Worker 0
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.58: Current state of the workers in the directory tree

ENQUEUE .../rootpath/initpath/path 2/path 5

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[1] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.59: Current code execution

path 5				

Figure 2.60: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- Worker 2
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- Worker 3
        hello.txt
      hello.txt <-- X
    path 3/ <-- Worker 0
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.61: Current state of the workers in the directory tree

DIR .../rootpath/initpath/path 2/path 5

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[0] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.62: Current code execution

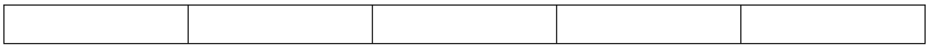


Figure 2.63: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- Worker 2
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- Worker 1
        hello.txt
      hello.txt <-- X
    path 3/ <-- Worker 0
      path 6/
        hello.txt
      hello.txt
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.64: Current state of the workers in the directory tree

ABSENT .../rootpath/initpath/path 3/hello.txt

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[0] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.65: Current code execution

Figure 2.66: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- Worker 2
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- Worker 1
        hello.txt
      hello.txt <-- X
    path 3/ <-- X
      path 6/
        hello.txt
      hello.txt <-- Worker 0
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.67: Current state of the workers in the directory tree

ENQUEUE .../rootpath/initpath/path 3/path 6

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[3] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.68: Current code execution

path 6				

Figure 2.69: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- Worker 2
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- Worker 1
        hello.txt
      hello.txt <-- X
    path 3/ <-- X
      path 6/ <-- Worker 0
        hello.txt
      hello.txt <-- X
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.70: Current state of the workers in the directory tree

DIR .../rootpath/initpath/path 3/path 6

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[2] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.71: Current code execution

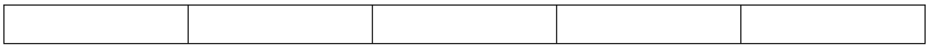


Figure 2.72: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- Worker 2
        path 7/
          hello.txt
        hello.txt
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- Worker 1
        hello.txt
      hello.txt <-- X
    path 3/ <-- X
      path 6/ <-- Worker 3
        hello.txt
      hello.txt <-- X
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.73: Current state of the workers in the directory tree

ABSENT .../rootpath/initpath/path 1/path 4/hello.txt

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[2] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.74: Current code execution

Figure 2.75: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- X
        path 7/
          hello.txt
        hello.txt <-- Worker 2
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- Worker 1
        hello.txt
      hello.txt <-- X
    path 3/ <-- X
      path 6/ <-- Worker 3
        hello.txt
      hello.txt <-- X
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.76: Current state of the workers in the directory tree

ENQUEUE .../rootpath/initpath/path 1/path 4/path 7

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[0] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.77: Current code execution

path 7				

Figure 2.78: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- X
        path 7/ <-- Worker 2
          hello.txt
        hello.txt <-- X
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- Worker 1
        hello.txt
      hello.txt <-- X
    path 3/ <-- X
      path 6/ <-- Worker 3
        hello.txt
      hello.txt <-- X
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.79: Current state of the workers in the directory tree

ENQUEUE .../rootpath/initpath/path 1/path 4/path 7

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[1] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.80: Current code execution

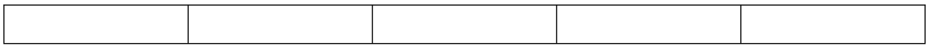


Figure 2.81: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- X
        path 7/ <-- Worker 0
          hello.txt
        hello.txt <-- X
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- Worker 1
        hello.txt
      hello.txt <-- X
    path 3/ <-- X
      path 6/ <-- Worker 3
        hello.txt
      hello.txt <-- X
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.82: Current state of the workers in the directory tree

ABSENT .../rootpath/initpath/path 2/path 5/hello.txt

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[3] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.83: Current code execution

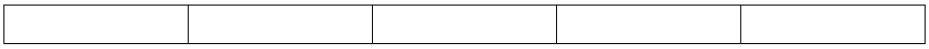


Figure 2.84: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- X
        path 7/ <-- Worker 0
          hello.txt
        hello.txt <-- X
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- X
        hello.txt <-- Worker 1
      hello.txt <-- X
    path 3/ <-- X
      path 6/ <-- Worker 3
        hello.txt
      hello.txt <-- X
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.85: Current state of the workers in the directory tree

ABSENT .../rootpath/initpath/path 3/path 6/hello.txt

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[0] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.86: Current code execution

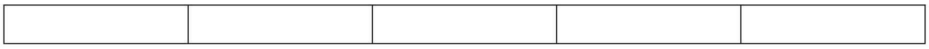


Figure 2.87: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- X
        path 7/ <-- Worker 0
          hello.txt
        hello.txt <-- X
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- X
        hello.txt <-- X
      hello.txt <-- X
    path 3/ <-- X
      path 6/ <-- X
        hello.txt <-- Worker 3
      hello.txt <-- X
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.88: Current state of the workers in the directory tree

```
PRESENT .../rootpath/initpath/path 1/path 4/path 7/hello.txt
```

```
[X] DIR /media/sf_cs140project2/rootpath
[X] ABSENT /media/sf_cs140project2/rootpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath
[X] DIR /media/sf_cs140project2/rootpath/initpath
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 2/path 5
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 3/path 6
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/hello.txt
[X] ENQUEUE /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[X] DIR /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 2/path 5/hello.txt
[X] ABSENT /media/sf_cs140project2/rootpath/initpath/path 3/path 6/hello.txt
[X] PRESENT /media/sf_cs140project2/rootpath/initpath/path 1/path 4/path 7/hello.txt
```

Figure 2.89: Current code execution

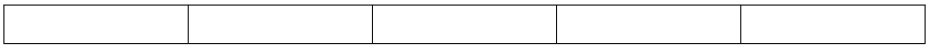


Figure 2.90: Current state of the task queue

```
rootpath/ <-- X
  initpath/ <-- X
    path 1/ <-- X
      path 4/ <-- X
        path 7/ <-- X
          hello.txt <-- Worker 0
        hello.txt <-- X
      hello.txt <-- X
    path 2/ <-- X
      path 5/ <-- X
        hello.txt <-- X
      hello.txt <-- X
    path 3/ <-- X
      path 6/ <-- X
        hello.txt <-- X
      hello.txt <-- X
    hello.txt <-- X
  hello.txt <-- X
single.c
multithreaded.c
```

Figure 2.91: Current state of the workers in the directory tree

(b) Explanation of how the task queue was implemented using your synchronization/IPC construct of choice; include how race conditions were handled.

Declaration of variables

First of all, we have to declare the variables that we need for our implementation.

```c
// Task queue implementation as a threadsafe linked list, with Lecture 16 and 18 from the CS 140 22.1 handlers as reference

// Synchronization primitives
sem_t c; // initialize the semaphore variable

// PATH as elem type
typedef char elem[MAX];

// Struct for the queue nodes
typedef struct __node_t {
    elem value;
    struct __node_t *next;
} node_t;

// Struct for the task queue
typedef struct __queue_t {
    node_t *head; // head of the queue
    node_t *tail; // tail of the queue
    pthread_mutex_t queue_lock; // initialize the queue lock
    int workers_sleeping; // count how many workers undergo sleeping
    int max_workers; // count the maximum amount of workers for the queue
} queue_t;

// Struct for the worker threads
typedef struct __worker {
    int wid; // worker ID
    elem string; // store the string to be searched on the globally shared queue
} worker;

// Declare the task queue
queue_t taskqueue;
```

Code Block 2.1: Declaration of necessary variables

- `sem_t c` defines the global semaphore treated as a "condition variable" for the processes. This will be initialized later when `main` gets executed.
- `char elem[MAX]` defines the `PATH` as a queue element.
- `struct __node_t` defines a queue node, which contains the value stored denoted by `elem value` and a link to the next element, denoted by `struct __node_t *next`.
- `struct __queue_t` defines the container for the task queue, containing the node pointers `node_t *head` and `node_t *tail` representing the front and rear end of the queue, respectively. It also contains a `pthread_mutex_t queue_lock` mutex lock for concurrency purposes, with two integers `int workers_sleeping` and `int max_workers` denoting the amount of workers to wait for work in the task queue and the maximum amount of workers deployed, respectively.

- struct __worker defines the characteristics of the worker, which in this case is a thread. It contains int wid for identity purposes and elem string which essentially is the search string the worker has to keep in mind for searching.

Queue functions

Now we go through the functions essential in queue management.

```c
// Initializing the task queue.
// This is run by the main thread.
void Queue_Init(queue_t *q, int workers_sleeping, int max_workers){
    node_t *tmp = malloc(sizeof(node_t));
    tmp->next = NULL;
    q->head = q->tail = tmp;
    q->workers_sleeping = workers_sleeping;
    q->max_workers = max_workers;
    pthread_mutex_init(&q->queue_lock, NULL);
    return;
}
```

Code Block 2.2: Initialization of the task queue

void Queue_Init initializes the queue upon call. It mainly works by allocating a starting queue node as its head and tail into the heap, such that the next element of the queue is set to NULL.

In addition, the task queue holds how many workers have returned, denoted by q->workers_sleeping, to find a task. The task queue also holds the total amount of workers deployed for string search, denoted by q->max_workers.

The line pthread_mutex_init(&q->queue_lock, NULL) is responsible for initializing the queue lock to avoid data races by deployed workers from queue accesses.

```c
// Threadsafe enqueueing in the task queue
void Queue_Enqueue(queue_t *q, elem value, int isSilent, worker * wkr){
    node_t *tmp = malloc(sizeof(node_t));
    snprintf(tmp->value, SNPRINTFMAX, "%s", value);
    tmp->next = NULL;
    pthread_mutex_lock(&q->queue_lock);
    q->tail->next = tmp;
    q->tail = tmp;
    pthread_mutex_unlock(&q->queue_lock);
    if (!isSilent) printf("[%d] ENQUEUE %s\n", wkr->wid, value);
    sem_post(&c);
    return;
}
```

Code Block 2.3: Enqueueing to the task queue

void Queue_Enqueue imitates the producer thread routine for enqueueing elements to the task queue. It starts by allocating a node to the heap and then setting the fields of the node based on the enqueued value. Note that this function has to call pthread_mutex_lock(&q->queue_lock) and pthread_mutex_unlock(&q->queue_lock) to lock the critical sections of setting the node values. After setting the fields successfully, it calls the printf statement for the element enqueued, and then signals a sleeping worker that called `sem_post(&c)`. If there are no sleeping threads, a thread that should undergo sleeping will not sleep because `sem_post(&c)` was called prior. The `isSilent` argument supresses the `ENQUEUE` output if necessary — this will be useful later when terminating the threads upon completion.

Note that `void Queue_Enqueue` is a candidate for data races due to the shared queue resource. Hence, to avoid data races from happening, we have to lock the queue from other workers while a workers acquires the lock for dequeue, as shown by `pthread_mutex_lock(&q->queue_lock)` and `pthread_mutex_unlock(&q->queue_lock)`.

```c
// Threadsafe dequeueing in the task queue
void Queue_Dequeue(queue_t *q, elem * value, worker * wkr){
    pthread_mutex_lock(&q->queue_lock);
    q->workers_sleeping++;
    while(q->head == q->tail){
        if (q->workers_sleeping == q->max_workers){
            // release the queue locks first
            pthread_mutex_unlock(&q->queue_lock);
            // this will call sem_post so the later sem_wait call will be negated
            thread_terminate(wkr);
            // reacquire the queue locks after enqueueing the thread termination signals
            pthread_mutex_lock(&q->queue_lock);
        }
        // release the queue locks first
        pthread_mutex_unlock(&q->queue_lock);
        sem_wait(&c);
        // reacquire the queue locks after waking up from sleep
        pthread_mutex_lock(&q->queue_lock);
    }
    q->workers_sleeping--;
    node_t *tmp = q->head;
    node_t *new_head  = tmp->next;
    snprintf(*value, SNPRINTFMAX, "%s", new_head->value);
    q->head = new_head;
    pthread_mutex_unlock(&q->queue_lock);
    if(strcmp(*value, "\n"))
        printf("[%d] DIR %s\n", wkr->wid, *value);
    free(tmp); // free the node
    return;
}

```

Code Block 2.4: Dequeueing from the task queue

`void Queue_Dequeue` imitates the `consumer` thread routine for dequeueing elements from the task queue. It starts by boldly assuming that a worker undergoes sleeping as denoted by `q->workers_sleeping++`. If the queue is empty, the worker should indeed enter the sleeping state, but first it must release the queue lock the thread currently holds. Once waken up, the worker should be running at this point, as denoted by `q->workers_sleeping--`.

However, if the queue is non-empty, the sleeping counter will be negated nevertheless. The worker proceeds on managing the queue first before taking its necessary task. Afterwards, the `printf` statement for `DIR` gets called, and then freeing the node that was allocated from earlier.

The `if (q->workers_sleeping == q->max_workers)` statement holds the key for terminating all the threads once there is no work left. More about this on a later section.

Note that void Queue_Dequeue is also a candidate for data races due to the shared queue resource. Hence, each worker that will dequeue from the queue has to acquire the lock and release it afterwards, as shown in pthread_mutex_lock(&q->queue_lock) and pthread_mutex_unlock(&q->queue_lock).

```c
// Threadsafe freeing the task queue
void Queue_Free(queue_t *q){
    node_t * tmp;
    while(q->head != NULL){
        tmp = q->head;
        q->head = q->head->next;
        free(tmp);
    }
    return;
}
```

Code Block 2.5: Freeing the task queue after use

void Queue_Free frees the queue nodes stored in the heap. It works by traversing the queue and freeing the nodes one-by-one. Freeing the queue essentially prevents memory leaks from happening.

The main thread

The main thread has to do its responsibilities on initializing the task queue and the deployment of workers before starting the search routine. The main thread is also responsible on parsing the arguments from the argv argument vector before passing them to the workers.

```c
    // Parse the number of workers N as an integer
    int n = atoi(argv[1]);
    // Initialize the threads
    pthread_t tid[n];
    // Initialize the task queue
    Queue_Init(&taskqueue, 0, n);
    // Initialize the semaphore for synchronization
    sem_init(&c, 0, 0);
    // Parse the rootpath
    char arg[MAX];
    // replace . or ./ with the absolute path of the working directory
    if (!(strcmp(argv[2], "."))||!(strcmp(argv[2], "./"))){
        char cwd[MAX];
        snprintf(arg, SNPRINTFMAX, "%s", getcwd(cwd,MAX));
    }
    // relative access - append the path with the working directory
    else if (argv[2][0] != '/') {
        char cwd[MAX];
        snprintf(arg, SNPRINTFMAX, "%s/%s", getcwd(cwd,MAX), argv[2]);
    }
    // if absolute path, stay as is
    else {
        snprintf(arg, SNPRINTFMAX, "%s", argv[2]);
    }
    // Enqueue the rootpath
    Queue_Enqueue(&taskqueue, arg, 1, 0); // enqueue the rootpath as first task
↪   without printing ENQUEUE
```

Code Block 2.6: Concurrent search preliminaries

The responsibilities of the main thread before the concurrent search are shown as comments in the provided code block.

Thread creation and join

The thread creation and join routines are shown by the following code block in `main`:

```c
    // ==================
    // CONCURRENT SECTION
    for (int i = 0; i < n; i++) {
        // store the worker ID on the heap
        worker * wkr = malloc(sizeof(worker));
        wkr->wid = i; // assign the worker ID to a worker
        snprintf(wkr->string, SNPRINTFMAX, "%s", argv[3]); // assign the string task to a worker
        pthread_create(&tid[i], NULL, f, wkr);
    }
    for (int i = 0; i < n; i++) {
        pthread_join(tid[i], NULL);
    }
    // ==================
```

Code Block 2.7: Thread create and join routines in `main`

Notice that before the threads are created using `pthread_create(&tid[i], NULL, f, wkr`, `main` has to assign the worker's identity and task first. Essentially the `main` thread is responsible on assigning the worker's identity before spawning, in which calls a worker function `f` denoting the worker's loop to find for tasks in the task queue.

After all workers are created, the `main` thread waits for the spawned workers to exit before waking up.

Worker function f

```c
// Worker function.
void *f(void * arg)
{
    // typecast void * to worker *
    worker * wkr = (worker *)arg;
    // initialize the path variable
    char path[MAX];
    // essentially an infinite loop until a thread exits
    while(1) {
        // Threads trying to dequeue will enter spinlock if a thread is dequeueing
        Queue_Dequeue(&taskqueue, &path, wkr);
        // THIS IS THREADSAFE - directories in the queue are disjoint
        dir_trav(path, wkr);
        // If the path obtained is a newline character, break from the loop. The newline character is the thread termination signal.
        if (!strcmp(path, "\n")) break;
    }
    // Terminate the remaining threads
    thread_terminate(wkr);
    free(wkr);
    return NULL;
}
```

Code Block 2.8: Thread create and join routines in `main`

Upon creation, each worker thread calls the worker function `f` that corresponds to its lifetime task. Until there is no work left, each worker follows the worker loop of the worker function. The worker loop has the following steps:

i. Dequeue the task queue if non-empty. Else, wait for a new task.

ii. Traverse the directory level assigned. Repeat.

Note that this worker loop is an infinite loop, which essentially only terminates if a sufficient condition is satisfied. In this case, the line `if (!strcmp(path, "\n")) break` corresponds to the thread termination routine within the loop. We will discuss this later.

If a worker already escapes the loop, it has to inform the other workers that the job was done, and it is time to terminate. The informant worker then frees its existence from the heap, and returns from the worker function.

Directory traversal

```
// THREADSAFE traversal of directory under path
void dir_trav(char *path, worker *wkr){
    DIR *dp; // directory pointer
    struct dirent *d; // directory entry
    if (!strcmp(path, "\n")) return;
    dp = opendir(path);
    if(dp == NULL){
        return; // empty directory
    }
    while ((d = readdir(dp)) != NULL){
        if (d->d_type == DT_DIR) {
            if (!(strcmp(d->d_name, "."))||!(strcmp(d->d_name, "..")))
                continue;
            char newpath[MAX];
            snprintf(newpath, SNPRINTFMAX, "%s/%s", path, d->d_name);
            Queue_Enqueue(&taskqueue, newpath, 0, wkr);
        }
        else if (d->d_type == DT_REG) {
            char command[MAX];
            snprintf(command, SNPRINTFMAX, "grep \"%s\" \"%s/%s\" > /dev/null",
↪   wkr->string, path, d->d_name);
            if (system(command) == 0) printf("[%d] PRESENT %s/%s\n", wkr->wid, path,
↪   d->d_name);
            else printf("[%d] ABSENT %s/%s\n", wkr->wid, path, d->d_name);
        }
    }
    closedir(dp);
    return;
}
```

Code Block 2.9: Directory traversal assigned to a worker

`void dir_trav` essentially is the function responsible for directory traversal. It takes the worker and path input as argument and essentially the worker looks at the path as reference for search.

Basically the worker opens up the `path` using the `opendir` command. If the returned directory pointer is `NULL`, immediately return from the function. This indicates that the directory level is empty, i.e. there are no directory entries nor files.

Otherwise, the `while ((d = readdir(dp)) != NULL)` loop gets called, essentially reading all existent directory entries `d` in the directory level pointed to by the directory pointer `dp`. If the read directory entry is also a directory, then enqueue the directory entry path to the task queue. Else, the worker executes `grep` using `system` while dumping the standard output to `/dev/null`. If the `system` command returns `0`, then the search string is present in the read file, printing the `PRESENT` statement using `printf`. Else, the search string is absent in the read file, printing the `ABSENT` statement using `printf`.

When there are no directory entries left in the directory level pointed to by the directory pointer `dp`, close the said directory pointer using the `closedir` command and return from the function.

Cleanup

The `main` thread has to cleanup the remnants of the string search. Basically, such remnants are the semaphore used for synchronization and the remnants of the task queue, in which the functions `sem_destroy(&c)` and `Queue_Free(&taskqueue)` are the ones responsible for the respective cleanups.

```
    sem_destroy(&c); // destroy the initialized semaphore
    // Free the taskqueue from the heap memory
    // At this point, the task queue MUST consist of ONE terminator task
    Queue_Free(&taskqueue);
```

Code Block 2.10: Program cleanup before exit

After cleanup, the `main` thread finally exits the program by the `exit(0)` command.

(c) Explanation of how each worker knows when to terminate (i.e., mechanism that determines and synchronizes that no more content will be enqueued); include how race conditions were handled.

The driver function for terminating threads is defined by the `thread_terminate` function.

```
// Thread termination routine
void thread_terminate(worker * wkr){
    char exitpath[MAX];
    snprintf(exitpath, SNPRINTFMAX, "\n");
    Queue_Enqueue(&taskqueue, exitpath, 1, wkr);
    return;
}
```

Code Block 2.11: Thread termination

Essentially this function holds the key for sending thread termination signals to other workers. What it basically does is just adding a newline path to the task queue, which sounds like an injection to the task queue.

The philosophy for this is that the workers only see the task queue and their specific tasks. The workers themselves have no idea about the existence of other workers so the thread termination signal has to enter the task queue which is a shared resource. This way the workers are able to at least communicate with each other knowing that their job is complete and it is time to terminate.

But how would the other workers know they are about to terminate? To do this, we have to determine how much work are left. We cannot have a predefined amount of work since a task is added to the task queue whenever a worker finds there is a directory free for traversal.

The solution for this is if the last worker finds itself to be the last thread awake, and it finds itself about to sleep because of an empty task queue. Note that whenever there are tasks available, there should be at least one worker awake who can enqueue new directories to the task queue. However, if that worker is about to dequeue and enter the sleeping state, essentially there is no work left to do, and hence that worker should escape from the sleeping state and enter the shutdown routine.

The start of the shutdown routine is actually shown earlier in the `void Queue_Dequeue` function, in which gets activated if the number of workers about to sleep is equal to the total number of deployed workers.

```
        if (q->workers_sleeping == q->max_workers){
            // release the queue locks first
            pthread_mutex_unlock(&q->queue_lock);
            // this will call sem_post so the later sem_wait call will be negated
            thread_terminate(wkr);
            // reacquire the queue locks after enqueueing the thread termination
 ↪  signals
            pthread_mutex_lock(&q->queue_lock);
        }
```

Code Block 2.12: Start of the shutdown routine

Note that since this routine is surrounded by the queue locks, we do not have to worry about data races. However, if we try to put this routine outside the locks, there is a chance that there would be data races intervening with the read data from the queue, causing the routine to not work as intended.

Note that before calling `thread_terminate` we have to unlock the queue lock — not doing this causes double locking from calling `void Queue_Enqueue` which renders the routine inaccessible.

After calling this routine, the terminator worker (i.e., the one who holds the termination signal to other workers) proceeds to get itself out of the worker loop. Since it now holds the termination signal denoted by the newline character `"\n"` as `PATH`, the terminator worker proceeds to satisfy the line `if (!strcmp(path, "\n")) break` to get out of the worker loop.

```
        if (!strcmp(path, "\n")) break;
    }
    // Terminate the remaining threads
    thread_terminate(wkr);
    free(wkr);
    return NULL;
}
```

Code Block 2.13: Getting out of the worker loop

Before the terminator worker exits, it calls `thread_terminate` for another time so that the rest of the threads still sleeping in `sem_wait(&c)` will be woken up. Essentially those threads will eventually realize that they are dequeueing a terminator signal, causing them to follow the termination routine as well.

After all the threads exit the worker function, one terminator signal will remain in the task queue. This remaining signal will eventually be removed using `void Queue_Free` anyways.

Bibliography

[1] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems : Three Easy Pieces*. Arpaci-Dusseau Books, 2014.

[2] Michael Kerrisk. *Linux man pages online*. URL: `https://man7.org/linux/man-pages/index.html`.

[3] E.P. Quiwa. *Data structures*. Electronics Hobbyists Publishing House, 2007. URL: `https://books.google.com.ph/books?id=LZCcswEACAAJ`.