

## 1.1)

A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, ( and a thread ID. )

Traditional ( heavyweight ) processes have a single thread of control - There is one program counter, and one sequence of instructions that can be carried out at any given time.

**Kernel-Level Threads:** To make concurrency cheaper, the execution aspect of process is separated out into threads. As such, the OS now manages threads and processes. All thread operations are implemented in the kernel and the OS schedules all threads in the system. OS managed threads are called kernel-level threads or light weight processes. NT: Threads Solaris: Lightweight processes (LWP).

In this method, the kernel knows about and manages the threads. No runtime system is needed in this case. Instead of thread table in each process, the kernel has a thread table that keeps track of all threads in the system. In addition, the kernel also maintains the traditional process table to keep track of processes. Operating Systems kernel provides system call to create and manage threads.

**advantages:-** Because kernel has full knowledge of all threads, Scheduler may decide to give more time to a process having large number of threads than process having small number of threads. Kernel-level threads are especially good for applications that frequently block.

**disadvantages:-** The kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads. Since kernel must manage and schedule threads as well as processes. It requires a full thread control block (TCB) for each thread to maintain information about threads. As a result there is significant overhead and increased in kernel complexity.

**User-Level Threads ;** Kernel-Level threads make concurrency much cheaper than process because, much less state to allocate and initialize. However, for fine-grained concurrency, kernel-level threads still suffer from too much overhead. Thread operations still require system calls. Ideally, we require thread operations to be as fast as a procedure call. Kernel-Level threads have to be general to support the needs of all programmers, languages, runtimes, etc. For such fine grained concurrency we need still "cheaper" threads.

To make threads cheap and fast, they need to be implemented at user level. User-Level threads are managed entirely by the run-time system (user-level library). The kernel knows nothing about user-level threads and manages them as if they were single-threaded processes. User-Level threads are small and fast, each thread is represented by a PC, register, stack, and small thread control block. Creating a new thread, switching between threads, and synchronizing threads are done via procedure call. i.e no kernel involvement. User-Level threads are hundred times faster than Kernel-Level threads.

**Advantages:-** The most obvious advantage of this technique is that a user-level threads package can be implemented on an Operating System that does not support threads. User-level threads does not require modification to operating systems. **Simple Representation:** Each thread is represented simply by a PC, registers, stack and a small control block, all stored in the user process address space. **Simple Management:** This simply means that creating a thread, switching between threads and synchronization between threads can all be done without intervention of the kernel. **Fast and Efficient:** Thread switching is not much more expensive than a procedure call.

**Disadvantages:-** User-Level threads are not a perfect solution as with everything else, they are a trade off. Since, User-Level threads are invisible to the OS they are not well integrated with the OS. As a result, OS can make poor decisions like scheduling a process with idle threads, blocking a process whose thread initiated an I/O even though the process has other threads that can run and unscheduling a process with a thread holding a lock. Solving this requires communication between kernel and user-level thread manager. There is a lack of coordination between threads and operating system kernel. Therefore, process as whole gets one time slice irrespective of whether process has one thread or 1000 threads within. It is up to each thread to relinquish control to other threads. User-level threads requires non-blocking systems call i.e., a multithreaded kernel. Otherwise, entire process will be blocked in the kernel, even if there are runnable threads left in the processes. For example, if one thread causes a page fault, the process blocks.

main difference is Users implement the user-level threads. On the other hand, the OS implements kernel-level threads.

User-level threads may be created and handled much faster. In contrast, kernel-level threads take longer to create and maintain.

The entire process is halted if a single user-level thread carries out a blocking operation. On the other hand, if a kernel thread carries out a blocking operation, another thread may continue to run. The user-level thread library includes the source code for thread creation, data transfer, thread destruction, message passing, and thread scheduling. On the other hand, the application code on kernel-level threads does not include thread management code. It is simply an API to the kernel mode. User-level threads do not invoke system calls for scheduling. On the other hand, system calls are used to generate and manage threads at the kernel level. The user-level thread is also referred to as the many-to-one mapping thread, as the OS assigns each thread in a multithreaded process to an execution context.

On the other hand, One-to-one thread mapping is supported at the kernel level. Each user thread must be assigned to a kernel thread. This mapping is handled by the OS. Context switch time is less in the user-level threads. On the other hand, context switch time is more in kernel-level threads. User-level threads may operate on any OS. In contrast, kernel-level threads are specific to the OS.

Multithread applications are unable to employ multiprocessing in user-level threads. In contrast, Kernel-level threads may be multithreaded.

Some instances of user-level threads are Java threads and POSIX threads. On the other hand, some instances of Kernel-level threads are Windows and Solaris.

**Conclusion** In summary, the primary distinction between User Level Threads and Kernel Level Threads is that the user maintains User Level Threads. In contrast, the kernel-level threads are handled by the OS. All modern OS supports the threading model, and the implementation of a thread will vary depending on the OS.

1.2.)

The screenshot shows a web browser window displaying the 'Count Word Frequency' application. The page has a light blue header with the title 'Count Word Frequency' and the subtitle 'cross-browser testing tools'. Below the header, there is a paragraph of text explaining the tool's purpose: 'World's simplest online word frequency calculator for web developers and programmers. Just paste your text in the form below, press the Calculate Word Frequency button, and you'll get individual word statistics. Press a button – get the word count. No ads, nonsense, or garbage.' A 'Like 51K' button is visible. An announcement states: 'Announcement: We just launched [math tools for developers](#). Check it out!'. Below this, a text area contains the following text: 'threads: 60', 'the: 48', 'a: 37', 'thread: 37', 'and: 30', 'to: 25', 'of: 22', 'is: 20', 'are: 20', 'in: 20', 'kernel: 20', 'process: 17', 'user-level: 17'. Below the text area are two buttons: 'Calculate Word Frequency' and 'Copy to clipboard (undo)'. At the bottom of the page, there is a prompt: 'Want to find the number of words in text? Use the [Word counter tool!](#)'. The browser's taskbar is visible at the bottom, showing various application icons and the system clock indicating 4:55 on 2/20/2.

threads: 60  
the: 48  
a: 37  
thread: 37  
and: 30  
to: 25  
of: 22  
is: 20  
are: 20  
in: 20  
kernel: 20  
process: 17  
user-level: 17  
kernel-level: 14

os: 13  
as: 12  
on: 11  
system: 10  
that: 9  
be: 9  
other: 9  
all: 8  
for: 8  
each: 7  
between: 7  
not: 7  
hand: 7  
this: 6  
user: 6  
processes: 5  
control: 5  
one: 5  
can: 5  
time: 5  
call: 5  
may: 5  
than: 5  
much: 5  
level: 5  
by: 5  
if: 5  
out: 4  
at: 4  
make: 4  
concurrency: 4  
operations: 4  
has: 4  
operating: 4  
small: 4  
block: 4  
require: 4  
they: 4  
an: 4  
with: 4  
contrast: 4  
there: 3  
manages: 3  
implemented: 3

about: 3  
table: 3  
manage: 3  
more: 3  
it: 3  
still: 3  
calls: 3  
fast: 3  
procedure: 3  
does: 3  
threaduser-level: 3  
simply: 3  
scheduling: 3  
blocking: 3  
multithreaded: 3  
handled: 3  
code: 3  
mapping: 3  
program: 2  
counter: 2  
stack: 2  
registers: 2  
(: 2  
): 2  
traditional: 2  
have: 2  
single: 2  
any: 2  
cheaper: 2  
execution: 2  
such: 2  
managed: 2  
or: 2  
knows: 2  
no: 2  
track: 2  
also: 2  
maintains: 2  
systems: 2  
create: 2  
full: 2  
having: 2  
number: 2  
applications: 2

times: 2  
must: 2  
well: 2  
maintain: 2  
result: 2  
overhead: 2  
less: 2  
we: 2  
support: 2  
need: 2  
represented: 2  
creating: 2  
done: 2  
faster: 2  
switching: 2  
even: 2  
requires: 2  
entire: 2  
will: 2  
carries: 2  
operation: 2  
context: 2  
switch: 2  
some: 2  
instances: 2  
1000: 1  
basic: 1  
unit: 1  
cpu: 1  
utilization: 1  
consisting: 1  
set: 1  
id: 1  
heavyweight: 1  
-: 1  
sequence: 1  
instructions: 1  
carried: 1  
given: 1  
threads:to: 1  
aspect: 1  
separated: 1  
into: 1  
now: 1

schedules: 1  
called: 1  
light: 1  
weight: 1  
processesnt:: 1  
threadssolaris:: 1  
lightweight: 1  
processes(lwp): 1  
method: 1  
runtime: 1  
needed: 1  
case: 1  
instead: 1  
keeps: 1  
addition: 1  
keep: 1  
provides: 1  
advantages:-because: 1  
knowledge: 1  
scheduler: 1  
decide: 1  
give: 1  
large: 1  
threadskernel-level: 1  
especially: 1  
good: 1  
frequently: 1  
diadvantages:-the: 1  
slow: 1  
inefficient: 1  
instance: 1  
hundreds: 1  
slower: 1  
threadssince: 1  
schedule: 1  
(tcb): 1  
information: 1  
significant: 1  
increased: 1  
complexity: 1  
;kernel-level: 1  
because: 1  
state: 1  
allocate: 1

initialize: 1  
however: 1  
fine-grained: 1  
suffer: 1  
from: 1  
too: 1  
ideally: 1  
general: 1  
needs: 1  
programmers: 1  
languages: 1  
runtimes: 1  
etc: 1  
fine: 1  
grained: 1  
"cheaper": 1  
cheap: 1  
entirely: 1  
run-time: 1  
(user-level: 1  
library)the: 1  
nothing: 1  
them: 1  
were: 1  
single-threaded: 1  
processesuser-level: 1  
pcregister,stack,: 1  
new: 1  
switching: 1  
synchronizing: 1  
via: 1  
ie: 1  
involvement: 1  
hundred: 1  
advantages:-the: 1  
most: 1  
obvious: 1  
advantage: 1  
technique: 1  
package: 1  
modification: 1  
systemssimple: 1  
representation:: 1  
pc: 1



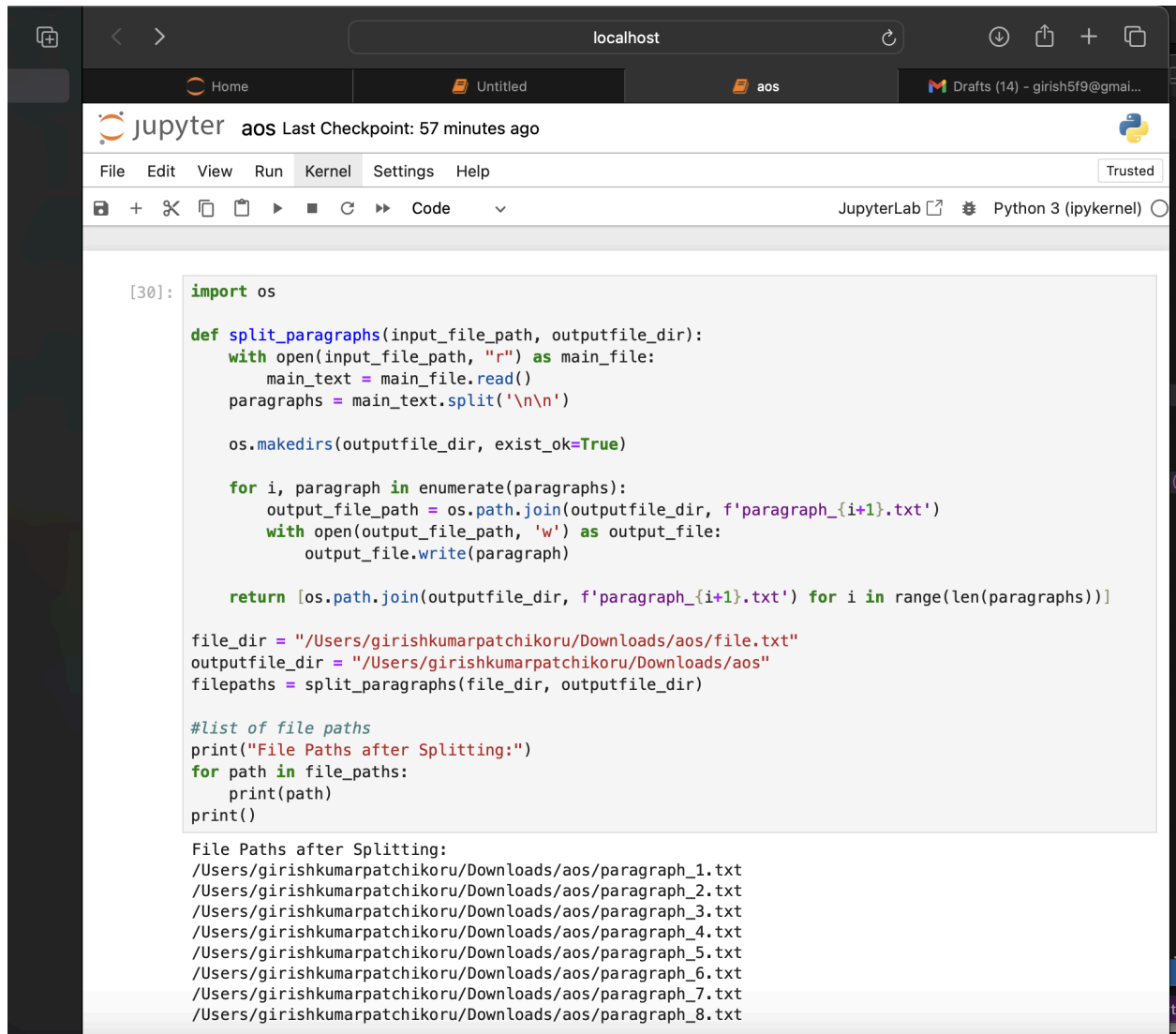
stored: 1  
address: 1  
spacesimple: 1  
management:: 1  
means: 1  
synchronization: 1  
without: 1  
intervention: 1  
kernelfast: 1  
efficient:: 1  
expensive: 1  
disadvantages:-: 1  
perfect: 1  
solution: 1  
everything: 1  
else: 1  
trade: 1  
off: 1  
since: 1  
invisible: 1  
integrated: 1  
poor: 1  
decisions: 1  
like: 1  
idle: 1  
whose: 1  
initiated: 1  
i/o: 1  
though: 1  
run: 1  
unscheduling: 1  
holding: 1  
lock: 1  
solving: 1  
communication: 1  
managerthere: 1  
lack: 1  
coordination: 1  
therefore: 1  
whole: 1  
gets: 1  
slice: 1  
irrespect: 1  
whether: 1

within: 1  
up: 1  
relinquish: 1  
non-blocking: 1  
ie.,: 1  
otherwise: 1  
blocked: 1  
runable: 1  
left: 1  
example: 1  
causes: 1  
page: 1  
fault: 1  
blocks: 1  
main: 1  
difference: 1  
users: 1  
implement: 1  
implements: 1  
created: 1  
take: 1  
longer: 1  
halted: 1  
another: 1  
continue: 1  
runthe: 1  
library: 1  
includes: 1  
source: 1  
creation: 1  
data: 1  
transfer: 1  
destruction: 1  
message: 1  
passing: 1  
application: 1  
include: 1  
management: 1  
api: 1  
modeuser-level: 1  
do: 1  
invoke: 1  
used: 1  
generate: 1

levelthe: 1  
referred: 1  
many-to-one: 1  
assigns: 1  
one-to-one: 1  
supported: 1  
assigned: 1  
oscontext: 1  
operate: 1  
specific: 1  
multithread: 1  
unable: 1  
employ: 1  
multiprocessing: 1  
java: 1  
posix: 1  
windows: 1  
solaris: 1  
conclusion: 1  
summary: 1  
primary: 1  
distinction: 1  
modern: 1  
supports: 1  
threading: 1  
model: 1  
implementation: 1  
vary: 1  
depending: 1

### 1.3)

Steps 1: The code establishes a function named "split\_paragraphs," which reads the content of a text file (file.txt) and divides it into paragraphs using double line breaks (\n\n) as delimiters. Subsequently, each paragraph is individually saved into distinct text files within the designated output directory (/Users/girishkumarpachikoru/Downloads/aos). These files are named sequentially as paragraph\_1.txt, paragraph\_2.txt, and so forth. Lastly, the code outputs a list of file paths that were generated as a result of the paragraph splitting process.



The screenshot shows a JupyterLab window with a browser address bar at 'localhost'. The interface includes a top bar with 'Home', 'Untitled', and 'aos' tabs, and a 'Drafts (14) - girish5f9@gmail...' notification. The JupyterLab logo and 'aos Last Checkpoint: 57 minutes ago' are visible. Below the menu bar (File, Edit, View, Run, Kernel, Settings, Help) is a toolbar with icons for file operations and a 'Code' dropdown. The main area displays a Python script in a code editor. The script defines a function 'split\_paragraphs' that takes an input file path and an output directory. It reads the input file, splits the text into paragraphs, creates the output directory, and then iterates over each paragraph to write it to a separate file. The script also defines file paths and prints the resulting list of file paths.

```
[30]: import os

def split_paragraphs(input_file_path, outputfile_dir):
    with open(input_file_path, "r") as main_file:
        main_text = main_file.read()
        paragraphs = main_text.split('\n\n')

    os.makedirs(outputfile_dir, exist_ok=True)

    for i, paragraph in enumerate(paragraphs):
        output_file_path = os.path.join(outputfile_dir, f'paragraph_{i+1}.txt')
        with open(output_file_path, 'w') as output_file:
            output_file.write(paragraph)

    return [os.path.join(outputfile_dir, f'paragraph_{i+1}.txt') for i in range(len(paragraphs))]

file_dir = "/Users/girishkumarpachikoru/Downloads/aos/file.txt"
outputfile_dir = "/Users/girishkumarpachikoru/Downloads/aos"
filepaths = split_paragraphs(file_dir, outputfile_dir)

#list of file paths
print("File Paths after Splitting:")
for path in filepaths:
    print(path)
print()

File Paths after Splitting:
/Users/girishkumarpachikoru/Downloads/aos/paragraph_1.txt
/Users/girishkumarpachikoru/Downloads/aos/paragraph_2.txt
/Users/girishkumarpachikoru/Downloads/aos/paragraph_3.txt
/Users/girishkumarpachikoru/Downloads/aos/paragraph_4.txt
/Users/girishkumarpachikoru/Downloads/aos/paragraph_5.txt
/Users/girishkumarpachikoru/Downloads/aos/paragraph_6.txt
/Users/girishkumarpachikoru/Downloads/aos/paragraph_7.txt
/Users/girishkumarpachikoru/Downloads/aos/paragraph_8.txt
```

## 2.1, 2.2 )Step 1:

During this initial phase, the code initiates the opening of the file specified by the `file\_path` in read mode ('r'), assigning its contents to the variable `text`. Subsequently, a list called `words` is generated by tokenizing the text into individual words, removing punctuation, and converting each word to lowercase. The resulting unique words are stored in the `unique\_words` variable.

## Step 2:

In this step, a dictionary named `word\_count\_dict` is established, with each unique word serving as a key and its corresponding value denoting the count of occurrences within the `words` list. This dictionary is then integrated into the `result\_dict` dictionary, utilizing the `file\_path` as the key and the word count dictionary as the associated value.

## Step 3:

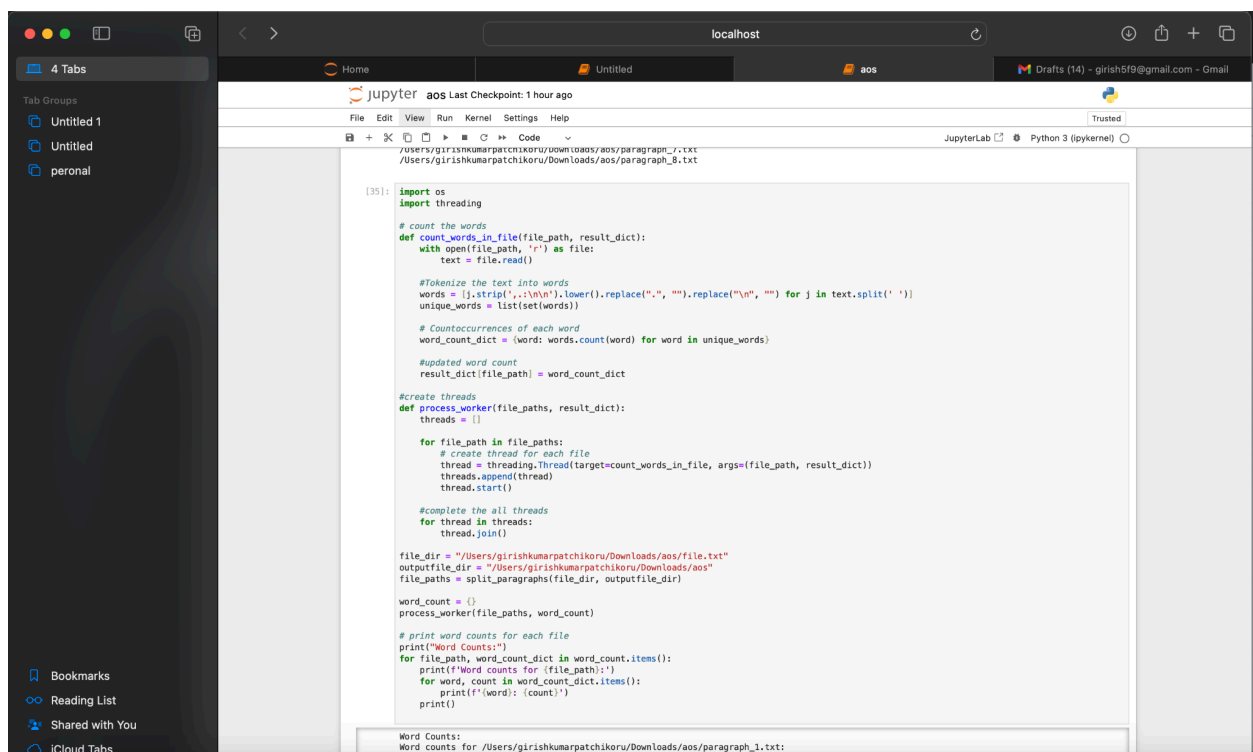
To enhance efficiency, individual threads are created for each file, employing the `threading.Thread` class. The `target` attribute is set to the `count\_words\_in\_file` function, and the `args` parameter facilitates the passage of `file\_path` and `result\_dict` to the function. These threads are appended to the `threads` list and subsequently activated.

for thread in threads:  
 thread.join()

The ensuing loop guarantees that the main program patiently awaits the completion of all threads by invoking the `join()` method on each. This sequential processing ensures the accurate calculation of word counts for each file before proceeding to subsequent steps.

Step 4:

Finally, the code traverses the `word\_count` dictionary, printing the word counts for each file. The internal loop is responsible for displaying each word alongside its count, and an additional `print()` statement introduces newline characters for improved readability between distinct files' word counts.



```
[35]: import os
import threading

# count the words
def count_words_in_file(file_path, result_dict):
    with open(file_path, 'r') as file:
        text = file.read()

    # Tokenize the text into words
    words = [i.strip('.,:;'\n').lower().replace("-", "").replace("'", "") for i in text.split(' ')]
    unique_words = list(set(words))

    # Count occurrences of each word
    word_count_dict = {word: words.count(word) for word in unique_words}

    # Update word count
    result_dict[file_path] = word_count_dict

# create threads
def process_worker(file_paths, result_dict):
    threads = []

    for file_path in file_paths:
        # create thread for each file
        thread = threading.Thread(target=count_words_in_file, args=(file_path, result_dict))
        threads.append(thread)
        thread.start()

    # complete the all threads
    for thread in threads:
        thread.join()

file_dir = "/Users/girishkumarpatchikoru/Downloads/aos/file.txt"
outputfile_dir = "/Users/girishkumarpatchikoru/Downloads/aos"
file_paths = split_paragraphs(file_dir, outputfile_dir)

word_count = {}
process_worker(file_paths, word_count)

# print word counts for each file
print("Word Counts:")
for file_path, word_count_dict in word_count.items():
    print(f"Word counts for {file_path}:")
    for word, count in word_count_dict.items():
        print(f'({word}: {count})')
    print()

Word Counts:
Word counts for /Users/girishkumarpatchikoru/Downloads/aos/paragraph_1.txt:
processes: 1
```

Word Counts:

Word counts for /Users/girishkumarpatchikoru/Downloads/aos/paragraph\_1.txt:  
processes: 1

carried: 1  
one: 2  
control: 1  
) : 1  
a: 7  
out: 1  
unit: 1  
registers: 1  
given: 1  
id: 1  
at: 1  
there: 1  
stack: 1  
single: 1  
heavyweight: 1  
)traditional: 1  
of: 5  
sequence: 1  
can: 1  
program: 2  
be: 1  
basic: 1  
counter: 2  
set: 1  
time: 1  
have: 1  
utilization: 1  
thread: 3  
and: 3  
cpu: 1  
any: 1  
( : 2  
- : 1  
that: 1  
instructions: 1  
consisting: 1  
is: 2

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_3.txt:

full: 2  
has: 1  
processes: 1  
maintain: 1  
must: 1  
operations: 1  
user-level: 1  
especially: 1  
control: 1  
overhead: 1

having: 2  
blockdiadvantages:-the: 1  
all: 1  
threads: 7  
to: 3  
number: 2  
information: 1  
a: 3  
increased: 1  
significant: 1  
there: 1  
more: 1  
complexity: 1  
in: 1  
good: 1  
process: 2  
(tcb): 1  
manage: 1  
well: 1  
each: 1  
kernel-level: 1  
times: 1  
frequently: 1  
small: 1  
inefficient: 1  
of: 5  
than: 2  
threadssince: 1  
slower: 1  
schedule: 1  
it: 1  
may: 1  
about: 1  
threadskernel-level: 1  
scheduler: 1  
instance: 1  
is: 1  
advantages:-because: 1  
are: 3  
time: 1  
large: 1  
thread: 2  
kernel: 3  
and: 3  
result: 1  
decide: 1  
hundreds: 1  
as: 3  
knowledge: 1

that: 2  
require: 1  
block: 1  
slow: 1  
for: 3  
applications: 1  
give: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_2.txt:

runtime: 1  
has: 1  
processes: 2  
aspect: 1  
operations: 1  
lightweight: 1  
also: 1  
separated: 1  
weight: 1  
track: 2  
all: 3  
threads: 8  
to: 2  
a: 1  
out: 1  
the: 11  
needed: 1  
keeps: 1  
concurrency: 1  
in: 6  
knows: 1  
process: 3  
into: 1  
implemented: 1  
manage: 1  
threads:to: 1  
processes(lwp)in: 1  
each: 1  
kernel-level: 2  
manages: 2  
no: 1  
of: 4  
operating: 1  
execution: 1  
this: 2  
or: 1  
method: 1  
about: 1  
make: 1  
maintains: 1



case: 1  
addition: 1  
call: 1  
provides: 1  
are: 2  
traditional: 1  
cheaper: 1  
managed: 1  
table: 3  
schedules: 1  
threadssolaris: 1  
thread: 3  
kernel: 5  
and: 4  
processesnt: 1  
instead: 1  
create: 1  
as: 1  
such: 1  
systems: 1  
light: 1  
that: 1  
keep: 1  
called: 1  
os: 3  
system: 4  
is: 2  
now: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_4.txt:

fine-grained: 1  
"cheaper": 1  
allocate: 1  
operations: 2  
fine: 1  
user-level: 1  
overhead: 1  
procedure: 1  
to: 4  
threads: 5  
all: 1  
runtimes: 1  
grained: 1  
a: 1  
;kernel-level: 1  
the: 1  
need: 1  
concurrency: 3  
process: 1

system: 1  
much: 3  
kernel-level: 2  
than: 1  
of: 1  
make: 1  
less: 1  
still: 3  
be: 2  
general: 1  
support: 1  
needs: 1  
etc: 1  
call: 1  
languages: 1  
initialize: 1  
cheaper: 1  
calls: 1  
ideally: 1  
have: 1  
thread: 2  
and: 1  
suffer: 1  
state: 1  
fast: 1  
we: 2  
as: 2  
because: 1  
programmers: 1  
such: 1  
too: 1  
require: 2  
however: 1  
for: 2  
from: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_5.txt:

: 6  
(user-level: 1  
they: 2  
pc,register,stack: 1  
entirely: 1  
block: 1  
user-level: 3  
nothing: 1  
control: 1  
procedure: 1  
to: 2  
threads: 8

done: 1  
library)the: 1  
a: 2  
switching: 1  
the: 1  
them: 1  
at: 1  
need: 1  
if: 1  
knows: 1  
hundred: 1  
implemented: 1  
represented: 1  
each: 1  
run-time: 1  
manages: 1  
ie: 1  
small: 2  
no: 1  
times: 1  
than: 1  
kernel-level: 1  
processesuser-level: 1  
new: 1  
cheap: 1  
level: 1  
about: 1  
make: 1  
creating: 1  
be: 1  
is: 1  
call: 1  
are: 4  
were: 1  
managed: 1  
synchronizing: 1  
via: 1  
between: 1  
thread: 3  
kernel: 2  
and: 5  
fast: 2  
faster: 1  
single-threaded: 1  
as: 1  
involvement: 1  
by: 2  
system: 1  
user: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_6.txt:

not: 3  
threaduser-level: 1  
does: 2  
: 2  
block: 1  
user-level: 1  
without: 1  
pc: 1  
control: 1  
user: 1  
spacesimple: 1  
obvious: 1  
threads: 4  
to: 1  
all: 2  
done: 1  
advantage: 1  
advantages:-the: 1  
a: 5  
synchronization: 1  
the: 2  
registers: 1  
procedure: 1  
package: 1  
stack: 1  
more: 1  
in: 1  
process: 1  
implemented: 1  
represented: 1  
systemssimple: 1  
kernelfast: 1  
much: 1  
each: 1  
small: 1  
of: 2  
operating: 2  
can: 2  
modification: 1  
switching: 2  
than: 1  
this: 2  
creating: 1  
intervention: 1  
be: 2  
on: 1  
support: 1

expensive: 1  
an: 1  
call: 1  
between: 2  
simply: 2  
thread: 3  
address: 1  
and: 3  
stored: 1  
management: 1  
technique: 1  
means: 1  
most: 1  
that: 3  
require: 1  
efficient: 1  
by: 1  
system: 1  
is: 3  
representation: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_7.txt:

1000: 1  
else: 1  
since: 1  
multithreaded: 1  
disadvantages:-: 1  
trade: 1  
a: 11  
the: 6  
managerthere: 1  
there: 1  
will: 1  
process: 8  
coordination: 1  
whole: 1  
fault: 1  
this: 1  
call: 1  
causes: 1  
are: 5  
idle: 1  
and: 3  
within: 1  
gets: 1  
lack: 1  
left: 1  
not: 2  
lock: 1

perfect: 1  
with: 4  
has: 2  
for: 1  
whose: 1  
processes: 1  
control: 1  
invisible: 1  
communication: 1  
up: 1  
runable: 1  
can: 2  
even: 2  
though: 1  
scheduling: 1  
between: 3  
result: 1  
blocks: 1  
solving: 1  
as: 3  
blocking: 1  
entire: 1  
they: 2  
everything: 1  
unscheduling: 1  
therefore: 1  
in: 2  
each: 1  
ie: 1  
operating: 1  
requires: 2  
integrated: 1  
make: 1  
like: 1  
time: 1  
run: 1  
systems: 1  
slice: 1  
os: 3  
system: 1  
initiated: 1  
threaduser-level: 1  
decisions: 1  
user-level: 3  
holding: 1  
one: 3  
to: 4  
threads: 8  
poor: 1

irrespect: 1  
solution: 1  
if: 2  
page: 1  
example: 1  
well: 1  
it: 1  
of: 2  
blocked: 1  
non-blocking: 1  
or: 1  
otherwise: 1  
relinquish: 1  
an: 1  
thread: 6  
kernel: 4  
off: 1  
that: 1  
i/o: 1  
whether: 1  
other: 2  
is: 2

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_8.txt:

: 1  
message: 1  
multithreaded: 1  
created: 1  
a: 5  
the: 12  
single: 1  
application: 1  
users: 1  
process: 2  
much: 1  
main: 1  
do: 1  
difference: 1  
may: 2  
implement: 1  
operation: 2  
are: 1  
simply: 1  
and: 4  
create: 1  
faster: 1  
generate: 1  
many-to-one: 1  
for: 2

not: 2  
assigns: 1  
contrast: 1  
longer: 1  
be: 1  
on: 5  
scheduling: 2  
calls: 2  
management: 1  
as: 2  
halted: 1  
handled: 1  
blocking: 2  
entire: 1  
invoke: 1  
does: 1  
destruction: 1  
out: 2  
api: 1  
at: 1  
in: 2  
passing: 1  
transfer: 1  
each: 1  
implements: 1  
take: 1  
maintainthe: 1  
data: 1  
used: 1  
code: 3  
os: 2  
modeuser-level: 1  
system: 2  
threadsuser-level: 1  
includes: 1  
source: 1  
user-level: 4  
also: 1  
to: 6  
threads: 6  
another: 1  
mapping: 1  
hand: 4  
carries: 2  
continue: 1  
if: 2  
context: 1  
creation: 1  
include: 1



manage: 1  
kernel-level: 3  
it: 1  
execution: 1  
levelthe: 1  
an: 2  
runthe: 1  
thread: 11  
kernel: 3  
referred: 1  
library: 1  
other: 4  
is: 4

### 3.2 final output and paste the report.



The screenshot shows a Jupyter Notebook interface. At the top, a status bar indicates 'is: 4'. Below it, a code cell is displayed with the following Python code:

```
[29]: def generate_final_report(word_count, output_file_path):  
    with open(output_file_path, 'w') as output_file:  
        for file_path, word_counts in word_count.items():  
            output_file.write(f'Word counts for {file_path}:\n')  
            for word, count in word_counts.items():  
                output_file.write(f'{word}: {count}\n')  
    print("Final Report Generated")  
  
# Example usage:  
output_file_path_2 = os.path.join(output_dir, 'result_file.txt')  
generate_final_report(word_count, output_file_path_2)
```

Below the code cell, the output 'Final Report Generated' is visible. At the bottom of the cell, there is an input prompt '[ ]:' followed by an empty text box.

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_1.txt:

processes: 1  
carried: 1  
one: 2  
control: 1  
) : 1  
a: 7  
out: 1  
unit: 1  
registers: 1

given: 1  
id: 1  
at: 1  
there: 1  
stack: 1  
single: 1  
heavyweight: 1  
)traditional: 1  
of: 5  
sequence: 1  
can: 1  
program: 2  
be: 1  
basic: 1  
counter: 2  
set: 1  
time: 1  
have: 1  
utilization: 1  
thread: 3  
and: 3  
cpu: 1  
any: 1  
(: 2  
-: 1  
that: 1  
instructions: 1  
consisting: 1  
is: 2

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_3.txt:

full: 2  
has: 1  
processes: 1  
maintain: 1  
must: 1  
operations: 1  
user-level: 1  
especially: 1  
control: 1  
overhead: 1  
having: 2  
blockdiadvantages:-the: 1  
all: 1  
threads: 7  
to: 3  
number: 2  
information: 1  
a: 3  
increased: 1

significant: 1  
there: 1  
more: 1  
complexity: 1  
in: 1  
good: 1  
process: 2  
(tcb): 1  
manage: 1  
well: 1  
each: 1  
kernel-level: 1  
times: 1  
frequently: 1  
small: 1  
inefficient: 1  
of: 5  
than: 2  
threadssince: 1  
slower: 1  
schedule: 1  
it: 1  
may: 1  
about: 1  
threadskernel-level: 1  
scheduler: 1  
instance: 1  
is: 1  
advantages:-because: 1  
are: 3  
time: 1  
large: 1  
thread: 2  
kernel: 3  
and: 3  
result: 1  
decide: 1  
hundreds: 1  
as: 3  
knowledge: 1  
that: 2  
require: 1  
block: 1  
slow: 1  
for: 3  
applications: 1  
give: 1  
Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_2.txt:  
runtime: 1

has: 1  
processes: 2  
aspect: 1  
operations: 1  
lightweight: 1  
also: 1  
separated: 1  
weight: 1  
track: 2  
all: 3  
threads: 8  
to: 2  
a: 1  
out: 1  
the: 11  
needed: 1  
keeps: 1  
concurrency: 1  
in: 6  
knows: 1  
process: 3  
into: 1  
implemented: 1  
manage: 1  
threads:to: 1  
processes(lwp)in: 1  
each: 1  
kernel-level: 2  
manages: 2  
no: 1  
of: 4  
operating: 1  
execution: 1  
this: 2  
or: 1  
method: 1  
about: 1  
make: 1  
maintains: 1  
case: 1  
addition: 1  
call: 1  
provides: 1  
are: 2  
traditional: 1  
cheaper: 1  
managed: 1  
table: 3  
schedules: 1

threadssolaris: 1

thread: 3

kernel: 5

and: 4

processesnt: 1

instead: 1

create: 1

as: 1

such: 1

systems: 1

light: 1

that: 1

keep: 1

called: 1

os: 3

system: 4

is: 2

now: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_4.txt:

fine-grained: 1

"cheaper": 1

allocate: 1

operations: 2

fine: 1

user-level: 1

overhead: 1

procedure: 1

to: 4

threads: 5

all: 1

runtimes: 1

grained: 1

a: 1

;kernel-level: 1

the: 1

need: 1

concurrency: 3

process: 1

system: 1

much: 3

kernel-level: 2

than: 1

of: 1

make: 1

less: 1

still: 3

be: 2

general: 1

support: 1

needs: 1  
etc: 1  
call: 1  
languages: 1  
initialize: 1  
cheaper: 1  
calls: 1  
ideally: 1  
have: 1  
thread: 2  
and: 1  
suffer: 1  
state: 1  
fast: 1  
we: 2  
as: 2  
because: 1  
programmers: 1  
such: 1  
too: 1  
require: 2  
however: 1  
for: 2  
from: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_5.txt:

: 6  
(user-level: 1  
they: 2  
pc,register,stack: 1  
entirely: 1  
block: 1  
user-level: 3  
nothing: 1  
control: 1  
procedure: 1  
to: 2  
threads: 8  
done: 1  
library)the: 1  
a: 2  
switching: 1  
the: 1  
them: 1  
at: 1  
need: 1  
if: 1  
knows: 1  
hundred: 1  
implemented: 1

represented: 1  
each: 1  
run-time: 1  
manages: 1  
ie: 1  
small: 2  
no: 1  
times: 1  
than: 1  
kernel-level: 1  
processesuser-level: 1  
new: 1  
cheap: 1  
level: 1  
about: 1  
make: 1  
creating: 1  
be: 1  
is: 1  
call: 1  
are: 4  
were: 1  
managed: 1  
synchronizing: 1  
via: 1  
between: 1  
thread: 3  
kernel: 2  
and: 5  
fast: 2  
faster: 1  
single-threaded: 1  
as: 1  
involvement: 1  
by: 2  
system: 1  
user: 1  
Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_6.txt:  
not: 3  
threadsuser-level: 1  
does: 2  
:  
block: 1  
user-level: 1  
without: 1  
pc: 1  
control: 1  
user: 1  
spacesimple: 1

obvious: 1  
threads: 4  
to: 1  
all: 2  
done: 1  
advantage: 1  
advantages:-the: 1  
a: 5  
synchronization: 1  
the: 2  
registers: 1  
procedure: 1  
package: 1  
stack: 1  
more: 1  
in: 1  
process: 1  
implemented: 1  
represented: 1  
systemssimple: 1  
kernelfast: 1  
much: 1  
each: 1  
small: 1  
of: 2  
operating: 2  
can: 2  
modification: 1  
switching: 2  
than: 1  
this: 2  
creating: 1  
intervention: 1  
be: 2  
on: 1  
support: 1  
expensive: 1  
an: 1  
call: 1  
between: 2  
simply: 2  
thread: 3  
address: 1  
and: 3  
stored: 1  
management: 1  
technique: 1  
means: 1  
most: 1



that: 3  
require: 1  
efficient: 1  
by: 1  
system: 1  
is: 3  
representation: 1  
Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_7.txt:  
1000: 1  
else: 1  
since: 1  
multithreaded: 1  
disadvantages:-: 1  
trade: 1  
a: 11  
the: 6  
managerthere: 1  
there: 1  
will: 1  
process: 8  
coordination: 1  
whole: 1  
fault: 1  
this: 1  
call: 1  
causes: 1  
are: 5  
idle: 1  
and: 3  
within: 1  
gets: 1  
lack: 1  
left: 1  
not: 2  
lock: 1  
perfect: 1  
with: 4  
has: 2  
for: 1  
whose: 1  
processes: 1  
control: 1  
invisible: 1  
communication: 1  
up: 1  
runable: 1  
can: 2  
even: 2  
though: 1

scheduling: 1  
between: 3  
result: 1  
blocks: 1  
solving: 1  
as: 3  
blocking: 1  
entire: 1  
they: 2  
everything: 1  
unscheduling: 1  
therefore: 1  
in: 2  
each: 1  
ie: 1  
operating: 1  
requires: 2  
integrated: 1  
make: 1  
like: 1  
time: 1  
run: 1  
systems: 1  
slice: 1  
os: 3  
system: 1  
initiated: 1  
threaduser-level: 1  
decisions: 1  
user-level: 3  
holding: 1  
one: 3  
to: 4  
threads: 8  
poor: 1  
irrespect: 1  
solution: 1  
if: 2  
page: 1  
example: 1  
well: 1  
it: 1  
of: 2  
blocked: 1  
non-blocking: 1  
or: 1  
otherwise: 1  
relinquish: 1  
an: 1

thread: 6  
kernel: 4  
off: 1  
that: 1  
i/o: 1  
whether: 1  
other: 2  
is: 2  
Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_8.txt:  
: 1  
message: 1  
multithreaded: 1  
created: 1  
a: 5  
the: 12  
single: 1  
application: 1  
users: 1  
process: 2  
much: 1  
main: 1  
do: 1  
difference: 1  
may: 2  
implement: 1  
operation: 2  
are: 1  
simply: 1  
and: 4  
create: 1  
faster: 1  
generate: 1  
many-to-one: 1  
for: 2  
not: 2  
assigns: 1  
contrast: 1  
longer: 1  
be: 1  
on: 5  
scheduling: 2  
calls: 2  
management: 1  
as: 2  
halted: 1  
handled: 1  
blocking: 2  
entire: 1  
invoke: 1

does: 1  
destruction: 1  
out: 2  
api: 1  
at: 1  
in: 2  
passing: 1  
transfer: 1  
each: 1  
implements: 1  
take: 1  
maintainthe: 1  
data: 1  
used: 1  
code: 3  
os: 2  
modeuser-level: 1  
system: 2  
threaduser-level: 1  
includes: 1  
source: 1  
user-level: 4  
also: 1  
to: 6  
threads: 6  
another: 1  
mapping: 1  
hand: 4  
carries: 2  
continue: 1  
if: 2  
context: 1  
creation: 1  
include: 1  
manage: 1  
kernel-level: 3  
it: 1  
execution: 1  
levelthe: 1  
an: 2  
runthe: 1  
thread: 11  
kernel: 3  
referred: 1  
library: 1  
other: 4  
is: 4

3.1)step 1:

Using a ThreadPoolExecutor, the code concurrently runs the `count\_words\_in\_file` function for each file path in the `file\_paths` list.

word\_count[file\_path] = word\_count\_dict

Upon concurrent processing of each file, the resulting word count dictionary (word\_count\_dict) for each file is stored in the global word\_count dictionary, with the file path serving as the key.

Step2:

for file\_path, word\_count\_dict in word\_count.items():

    print(f'Word counts for {file\_path}:')

    for word, count in word\_count\_dict.items():

        print(f'{word}: {count}')

    print()

the code iterates through the word\_count dictionary and prints the combined word counts for each file. The inner loop displays each word alongside its count, and an additional print() statement introduces newline characters for improved readability between the word counts of different files.

```
[12]: from concurrent.futures import ThreadPoolExecutor

word_count = {}

def count_words_in_file(file_path):
    with open(file_path, 'r') as file:
        text = file.read()

    words = [j.strip(',.\n\n').lower().replace(".", "").replace("\n", "") for j in text.split(' ')]
    unique_words = list(set(words))

    word_count_dict = {word: words.count(word) for word in unique_words}

    # Store the result variable word_count
    word_count[file_path] = word_count_dict

def process_worker(file_paths):
    with ThreadPoolExecutor() as executor:
        executor.map(count_words_in_file, file_paths)

file_dir = "/Users/girishkumarpatchikoru/Downloads/aos/file.txt"
outputfile_dir = "/Users/girishkumarpatchikoru/Downloads/aos"
file_paths = split_paragraphs(file_dir, outputfile_dir)

# call functions
process_worker(file_paths)

# print the aggregated word counts
print("Aggregated Word Counts:")
for file_path, word_count_dict in global_word_count.items():
    print(f'Word counts for {file_path}:')
    for word, count in word_count_dict.items():
        print(f'{word}: {count}')
    print()
```

Aggregated Word Counts:  
Word counts for /Users/girishkumarpatchikoru/Downloads/aos/paragraph\_1.txt:  
control: 1

Aggregated Word Counts:

Word counts for /Users/girishkumarpatchikoru/Downloads/aos/paragraph\_1.txt:

control: 1

have: 1

basic: 1

)traditional: 1

unit: 1

that: 1

registers: 1  
carried: 1  
sequence: 1  
given: 1  
(: 2  
can: 1  
): 1  
single: 1  
there: 1  
counter: 2  
a: 7  
any: 1  
set: 1  
program: 2  
is: 2  
time: 1  
processes: 1  
stack: 1  
cpu: 1  
id: 1  
instructions: 1  
of: 5  
thread: 3  
heavyweight: 1  
one: 2  
-: 1  
be: 1  
and: 3  
at: 1  
utilization: 1  
out: 1  
consisting: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_3.txt:

it: 1  
times: 1  
control: 1  
number: 2  
require: 1  
(tcb): 1  
especially: 1  
that: 2  
inefficient: 1  
give: 1  
more: 1  
full: 2  
threadssince: 1  
information: 1  
well: 1

as: 3  
small: 1  
are: 3  
all: 1  
maintain: 1  
to: 3  
applications: 1  
kernel-level: 1  
there: 1  
may: 1  
process: 2  
a: 3  
overhead: 1  
instance: 1  
schedule: 1  
slower: 1  
in: 1  
complexity: 1  
knowledge: 1  
manage: 1  
is: 1  
time: 1  
result: 1  
processes: 1  
advantages:-because: 1  
than: 2  
frequently: 1  
operations: 1  
hundreds: 1  
about: 1  
good: 1  
each: 1  
decide: 1  
has: 1  
scheduler: 1  
user-level: 1  
threads: 7  
of: 5  
blockdiadvantages:-the: 1  
threadskernel-level: 1  
must: 1  
thread: 2  
slow: 1  
significant: 1  
kernel: 3  
and: 3  
block: 1  
increased: 1  
having: 2

for: 3  
large: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_2.txt:

os: 3  
or: 1  
that: 1  
systems: 1  
such: 1  
threads:to: 1  
called: 1  
into: 1  
processes(lwp)in: 1  
table: 3  
keeps: 1  
threadssolaris: 1  
cheaper: 1  
case: 1  
traditional: 1  
make: 1  
separated: 1  
as: 1  
needed: 1  
manages: 2  
are: 2  
all: 3  
weight: 1  
concurrency: 1  
to: 2  
keep: 1  
kernel-level: 2  
knows: 1  
track: 2  
system: 4  
no: 1  
process: 3  
a: 1  
in: 6  
schedules: 1  
instead: 1  
create: 1  
manage: 1  
is: 2  
aspect: 1  
operating: 1  
execution: 1  
maintains: 1  
processes: 2  
the: 11



method: 1  
operations: 1  
implemented: 1  
about: 1  
now: 1  
also: 1  
each: 1  
managed: 1  
addition: 1  
has: 1  
threads: 8  
of: 4  
thread: 3  
light: 1  
lightweight: 1  
runtime: 1  
call: 1  
this: 2  
kernel: 5  
and: 4  
out: 1  
processesnt: 1  
provides: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_4.txt:

have: 1  
from: 1  
because: 1  
allocate: 1  
require: 2  
we: 2  
initialize: 1  
procedure: 1  
such: 1  
needs: 1  
cheaper: 1  
calls: 1  
make: 1  
suffer: 1  
languages: 1  
much: 3  
state: 1  
as: 2  
all: 1  
to: 4  
concurrency: 3  
support: 1  
kernel-level: 2  
system: 1

process: 1  
overhead: 1  
still: 3  
a: 1  
fast: 1  
less: 1  
fine-grained: 1  
too: 1  
general: 1  
fine: 1  
programmers: 1  
than: 1  
the: 1  
runtimes: 1  
operations: 2  
;kernel-level: 1  
etc: 1  
"cheaper": 1  
user-level: 1  
threads: 5  
of: 1  
thread: 2  
need: 1  
call: 1  
be: 2  
and: 1  
however: 1  
ideally: 1  
grained: 1  
for: 2

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_6.txt:

: 2  
control: 1  
require: 1  
stored: 1  
simply: 2  
synchronization: 1  
that: 3  
registers: 1  
procedure: 1  
more: 1  
obvious: 1  
represented: 1  
on: 1  
expensive: 1  
much: 1  
small: 1  
by: 1

all: 2  
to: 1  
pc: 1  
support: 1  
means: 1  
done: 1  
creating: 1  
without: 1  
does: 2  
spacesimple: 1  
system: 1  
kernelfast: 1  
process: 1  
a: 5  
representation: 1  
switching: 2  
systemssimple: 1  
intervention: 1  
in: 1  
management: 1  
threaduser-level: 1  
advantages:-the: 1  
is: 3  
an: 1  
modification: 1  
operating: 2  
the: 2  
than: 1  
implemented: 1  
most: 1  
stack: 1  
each: 1  
advantage: 1  
between: 2  
user-level: 1  
technique: 1  
threads: 4  
of: 2  
package: 1  
thread: 3  
call: 1  
this: 2  
be: 2  
not: 3  
and: 3  
block: 1  
address: 1  
efficient: 1  
can: 2

user: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_7.txt:

it: 1

os: 3

causes: 1

managerthere: 1

make: 1

scheduling: 1

as: 3

can: 2

are: 5

up: 1

fault: 1

disadvantages:-: 1

initiated: 1

solution: 1

integrated: 1

communication: 1

unscheduling: 1

other: 2

processes: 1

run: 1

though: 1

call: 1

this: 1

whose: 1

and: 3

decisions: 1

gets: 1

entire: 1

with: 4

requires: 2

i/o: 1

invisible: 1

idle: 1

process: 8

time: 1

operating: 1

blocked: 1

irrespect: 1

each: 1

since: 1

perfect: 1

like: 1

user-level: 3

has: 2

threads: 8

of: 2

thread: 6  
blocks: 1  
not: 2  
ie: 1  
solving: 1  
everything: 1  
that: 1  
trade: 1  
holding: 1  
else: 1  
well: 1  
otherwise: 1  
blocking: 1  
left: 1  
off: 1  
system: 1  
a: 11  
is: 2  
poor: 1  
multithreaded: 1  
between: 3  
even: 2  
kernel: 4  
for: 1  
control: 1  
lock: 1  
or: 1  
systems: 1  
if: 2  
whether: 1  
whole: 1  
slice: 1  
within: 1  
therefore: 1  
to: 4  
there: 1  
page: 1  
in: 2  
threaduser-level: 1  
an: 1  
runable: 1  
result: 1  
will: 1  
the: 6  
coordination: 1  
non-blocking: 1  
1000: 1  
one: 3  
example: 1

they: 2  
relinquish: 1  
lack: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_8.txt:

: 1  
it: 1  
os: 2  
implement: 1  
maintainthe: 1  
destruction: 1  
scheduling: 2  
many-to-one: 1  
as: 2  
modeuser-level: 1  
are: 1  
does: 1  
create: 1  
execution: 1  
other: 4  
operation: 2  
api: 1  
continue: 1  
and: 4  
assigns: 1  
created: 1  
entire: 1  
halted: 1  
on: 5  
calls: 2  
much: 1  
single: 1  
process: 2  
includes: 1  
manage: 1  
used: 1  
take: 1  
users: 1  
runthe: 1  
also: 1  
generate: 1  
each: 1  
user-level: 4  
threads: 6  
implements: 1  
thread: 11  
handled: 1  
carries: 2  
not: 2

at: 1  
application: 1  
simply: 1  
source: 1  
blocking: 2  
hand: 4  
may: 2  
referred: 1  
system: 2  
a: 5  
data: 1  
transfer: 1  
faster: 1  
is: 4  
main: 1  
multithreaded: 1  
invoke: 1  
code: 3  
longer: 1  
creation: 1  
another: 1  
passing: 1  
be: 1  
kernel: 3  
for: 2  
out: 2  
if: 2  
include: 1  
to: 6  
kernel-level: 3  
levelthe: 1  
message: 1  
mapping: 1  
in: 2  
management: 1  
threaduser-level: 1  
an: 2  
contrast: 1  
context: 1  
the: 12  
do: 1  
difference: 1  
library: 1

Word counts for /Users/girishkumarpachikoru/Downloads/aos/paragraph\_5.txt:

: 6  
control: 1  
times: 1  
processesuser-level: 1

entirely: 1  
via: 1  
procedure: 1  
represented: 1  
if: 1  
were: 1  
level: 1  
make: 1  
switching: 1  
them: 1  
as: 1  
cheap: 1  
are: 4  
by: 2  
manages: 1  
nothing: 1  
to: 2  
small: 2  
done: 1  
run-time: 1  
knows: 1  
creating: 1  
kernel-level: 1  
system: 1  
no: 1  
(user-level: 1  
a: 2  
fast: 2  
hundred: 1  
faster: 1  
is: 1  
the: 1  
than: 1  
implemented: 1  
about: 1  
each: 1  
new: 1  
managed: 1  
between: 1  
synchronizing: 1  
user-level: 3  
library)the: 1  
threads: 8  
thread: 3  
pc,register,stack: 1  
need: 1  
call: 1  
involvement: 1  
be: 1



kernel: 2  
and: 5  
at: 1  
they: 2  
block: 1  
single-threaded: 1  
ie: 1  
user: 1