```python
import os
import threading
import requests
import logging
import argparse
import hashlib
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry
from time import time
from bs4 import BeautifulSoup  # Optional for additional processing

# ========== Logging Setup ==========
logging.basicConfig(level=logging.INFO, format='[%(levelname)s] %(asctime)s - %(message)s')

# ========== Helper Functions ==========
def hash_url(url: str) -> str:
    return hashlib.md5(url.encode()).hexdigest()

def cache_exists(cache_dir: str, url_hash: str) -> bool:
    return os.path.exists(os.path.join(cache_dir, f"{url_hash}.html"))

def save_to_cache(cache_dir: str, url_hash: str, content: str):
    os.makedirs(cache_dir, exist_ok=True)
    with open(os.path.join(cache_dir, f"{url_hash}.html"), "w", encoding="utf-8") as f:
        f.write(content)

# ========== WebPageDownloader Class ==========
class WebPageDownloader:
    def __init__(self, user_agent=None, cache_dir=".cache", timeout=10):
        self.session = requests.Session()
        self.timeout = timeout
        self.cache_dir = cache_dir

        retries = Retry(total=5, backoff_factor=1, status_forcelist=[500, 502, 503, 504])
        self.session.mount("http://", HTTPAdapter(max_retries=retries))
        self.session.mount("https://", HTTPAdapter(max_retries=retries))

        self.headers = {
            "User-Agent": user_agent or "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
        }

    def fetch(self, url: str) -> str:
        url_hash = hash_url(url)
        if cache_exists(self.cache_dir, url_hash):
            logging.info(f"[CACHE] Using cached version for {url}")
            with open(os.path.join(self.cache_dir, f"{url_hash}.html"), "r",
encoding="utf-8") as f:
                return f.read()

        logging.info(f"[FETCH] Requesting {url}")
        try:
            response = self.session.get(url, headers=self.headers,
timeout=self.timeout)
            response.raise_for_status()
            save_to_cache(self.cache_dir, url_hash, response.text)
            return response.text
        except requests.exceptions.RequestException as e:
            logging.error(f"[ERROR] Failed to fetch {url}: {e}")
            return ""

    def save(self, content: str, output_file: str):
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(content)
        logging.info(f"[SAVE] Content saved to {output_file}")

# ========== Threaded Execution ==========
def download_and_save(url: str, output_file: str, downloader: WebPageDownloader):
    content = downloader.fetch(url)
    if content:
        downloader.save(content, output_file)

# ========== Main ==========
def main():
    parser = argparse.ArgumentParser(description="Download webpages in parallel and
cache results.")
    parser.add_argument("urls", nargs="+", help="One or more URLs to download")
    parser.add_argument("-o", "--outdir", default="outputs", help="Directory to save
downloaded files")
    parser.add_argument("--threads", type=int, default=4, help="Number of concurrent
threads")
    args = parser.parse_args()

    os.makedirs(args.outdir, exist_ok=True)
    downloader = WebPageDownloader()

    threads = []
    for url in args.urls:
        filename = hash_url(url)[:10] + ".html"
        filepath = os.path.join(args.outdir, filename)
        thread = threading.Thread(target=download_and_save, args=(url, filepath,
downloader))
        thread.start()
        threads.append(thread)

    for t in threads:
        t.join()

    logging.info("All downloads completed.")

if __name__ == "__main__":
    main()
```