# Operating Systems - Assignment 4: Parallel sort using pthreads

Winter 2025

## Assignment Specification

In this assignment, you will implement in C a program called `psort`, which sorts fixed-size binary records in parallel.

Usage:

```
./psort input_file output_file
```

Input file format:

- A sequence of 100-byte records, one after another.
- The first 4 bytes of each record is the key that you will use for sorting (these 4 bytes are included within the mentioned 100 bytes).
- You can assume the entire file fits into memory.

Output file format and requirements:

- Same format: a sequence of 100-bytes per record.
- Before exiting, your program must call `fsync()` to flush all data to disk.

## Requirements

Your program must:

1. Create multiple threads using the `pthread.h` library.
2. Use `get_nprocs()` to detect how many CPUs are available, and spawn that many threads.
3. Read the input file into memory (use `mmap()`).
4. Divide the work among the threads (e.g., by splitting the data range).
5. Sort records in parallel.
6. Merge-sort chunks into the final output.
7. Write the final sorted output to the specified file.

## Implementation Notes

- You may use `qsort()` from `stdlib.h` for single-threaded sorting in chunks.
- Use locks only where needed; avoid unnecessary contention.
- Avoid global variables unless protected with synchronization.
- Memory mapping the input file (`mmap()`) is encouraged but not required.
- Merging the sorted subarrays is a serial phase; keep it efficient. Consider using heap-based k-way merge.

## Compilation

Your code must compile using the following command:

```
gcc -Wall -Werror -pthread -O -o psort psort.c
```

Use the `-O` optimization flag. Try comparing runtimes with and without it and include the results in your report.

## Random file generator

Find on moodle the program `gen.c` that generates random input files for your program. Compile it, and to generate, for instance, an input file with 1 million entries, run:

```
./gen sortme.bin 1000000
```

The generated file will be a little less than 100 MBytes.

Record timing for large input files using `time`; write the timing results in your report.

## Deliverables

Submit a `.zip` file named `hw4_<student_id>.zip` containing:

- Your source code (`psort.c` and any helper files).
- A short PDF report (`report_<student_id>.pdf`) that follows this structure:
  1. Your name and ID.
  2. A brief explanation of your implementation strategy; mention all the functions and system calls that you used.
  3. How you divided the work across threads.
  4. How you tested correctness (you may provide a second C program to check correctness).
  5. How you measured performance.
  6. Any known issues or limitations.

## Evaluation

- Your program will be evaluated for correctness mostly, and it needs to have an acceptable level or performance.
- If there are obvious performance problems, you may lose points, so make sure your program is actually running faster than a single-threaded program doing the same task.

## Checklist Before Submitting

- Program compiles with `-Wall -Werror -pthread`
- Program uses number of cores detected via `get_nprocs()`
- Output file is correctly sorted
- Output file size matches the input
- `fsync()` is called before the program exits
- The report is complete
- The `.zip` file contains all required files