Nagar Yuwak Shikshan Sanstha's
Yeshwantrao Chavan College of Engineering
(An Autonomous Institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)
Hingna Road, Wanadongri, Nagpur - 441 110
NAAC A++
Ph.: 07104-237919, 234623, 329249, 329250 Fax: 07104-232376, Website: www.ycce.edu

## Department of Computer Technology

**Vision of the Department**

*To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.*

**Mission of the Department**

*To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.*

**Session 2025-2026**

| | |
|---|---|
| **Vision:** To harness the power of artificial intelligence and data science to solve real-world problems and enhance human potential. | **Mission:** To acquire skills through coursework, projects, and internships, while actively engaging in research and collaboration with peers to innovate and apply AI solutions. |

**Program Educational Objectives of the program (PEO):** (broad statements that describe the professional and career accomplishments)

| | | | |
|---|---|---|---|
| PEO1 | **Preparation** | **P: Preparation** | **Pep-CL abbreviation pronounce as Pep-si-lL easy to recall** |
| PEO2 | **Core Competence** | **E: Environment (Learning Environment)** | |
| PEO3 | **Breadth** | **P: Professionalism** | |
| PEO4 | **Professionalism** | **C: Core Competence** | |
| PEO5 | **Learning Environment** | **L: Breadth (Learning in diverse areas)** | |

**Program Outcomes (PO):** (statements that describe what a student should be able to do and know by the end of a program)

**Keywords of POs:**
Engineering knowledge, Problem analysis, Design/development of solutions, Conduct Investigations of Complex Problems, Engineering Tool Usage, The Engineer and The World, Ethics, Individual and Collaborative Team work, Communication, Project Management and Finance, Life-Long Learning

**PSO Keywords:** Cutting edge technologies, Research

"I am an engineer, and I know how to apply engineering knowledge to investigate, analyse and design solutions to complex problems using tools for entire world following all ethics in a collaborative way with proper management skills throughout my life." *to contribute to the development of cutting-edge technologies and Research*.

**Integrity:** I will adhere to the Laboratory Code of Conduct and ethics in its entirety.

Prerana Bijekar
Purvaja Sawalakhe 1 November 2025
**Name and Signature of Student and Date**
(Signature and Date in Handwritten)

Nagar Yuwak Shikshan Sanstha's
Yeshwantrao Chavan College of Engineering
(An Autonomous Institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)
Hingna Road, Wanadongri, Nagpur - 441 110
NAAC A++
Ph.: 07104-237919, 234623, 329249, 329250 Fax: 07104-232376, Website: www.ycce.edu
Department of Computer Technology

**Vision of the Department**

*To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.*

**Mission of the Department**

*To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.*

| | | | |
|---|---|---|---|
| **Session** | 2025-26 (ODD) | **Course Name** | HPC Lab |
| **Semester** | 7 | **Course Code** | 22ADS706 |
| **Roll No.** | 11,<br>12 | **Name of Students** | Prerana Bijekar,<br>Purvaja Sawalakhe |

| | |
|---|---|
| Practical No. | 9 |
| Course Outcome | **CO1:** Understand and Apply Parallel Programming Concepts<br>**CO2:** Analyze and Improve Program Performance.<br>**CO3:** Demonstrate Practical Skills in HPC Tools and Environments. |
| Aim | Process vs Thread Performance |
| Theory (100 words) | A process is an independent program in execution with its own memory space, while a thread is a lightweight subprocess that shares memory and resources within a process. Processes provide isolation and stability, but creating and switching between them incurs higher overhead due to context switching and memory duplication. Threads, on the other hand, allow faster communication and execution within the same address space, making them more efficient for concurrent tasks. Performance comparison between processes and threads helps determine which model is more suitable for a given workload—CPU-bound tasks may benefit from multiple processes, whereas I/O-bound or parallelizable tasks often perform better with threads. |
| Procedure and Execution (100 Words) | Steps of implementation:<br>1. Write two programs — one using multiple processes and another using multiple threads to perform the same computational task (e.g., summing array elements or calculating factorials).<br>2. Use libraries such as multiprocessing and threading in Python (or equivalent in other languages).<br>3. Measure execution time for both implementations using time functions.<br>4. Record CPU usage and memory consumption.<br>5. Compare and analyze results to determine performance differences. |
| | Algorithm:<br>1. Start the program and initialize the dataset.<br>2. Record the start time.<br>3. Create multiple processes/threads to perform the same computation concurrently.<br>4. Wait for all processes/threads to complete.<br>5. Record the end time and compute total execution time.<br>6. Display or store the time and resource usage for comparison. |

**Vision of the Department**

*To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.*

**Mission of the Department**

*To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.*

Code:

```
[lab1@localhost ~]$ mkdir ~/process_thread_project
[lab1@localhost ~]$ cd ~/process_thread_project
[lab1@localhost process_thread_project]$ nano process_thread_perf.c
[lab1@localhost process_thread_project]$ nano Makefile
[lab1@localhost process_thread_project]$ gcc -O2 -Wall -Wextra -pthread -o process_thread_perf process_thread_perf.c
process_thread_perf.c: In function 'run_threads':
process_thread_perf.c:96:32: warning: unused parameter 'mode' [-Wunused-parameter]
   96 | double run_threads(const char *mode, int workers, long iterations, const char *filename) {
      |                    ~~~~~~~~~~~~^~~~
process_thread_perf.c: In function 'run_processes':
process_thread_perf.c:128:34: warning: unused parameter 'mode' [-Wunused-parameter]
  128 | double run_processes(const char *mode, int workers, long iterations, const char *filename) {
      |                      ~~~~~~~~~~~~^~~~
```

GNU nano 5.6.1

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <pthread.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>

typedef struct {
    long iterations;
    const char *filename;
    int id;
} worker_arg_t;

static inline double timespec_to_sec(const struct timespec *t) {
    return t->tv_sec + t->tv_nsec / 1e9;
}

/* Simple CPU-bound workload: do floating-point ops in a loop */
void cpu_work(long iterations) {
    volatile double x = 0.123456;
    for (long i = 0; i < iterations; ++i) {
        /* some math that is hard to optimize away */
        x += (i % 7) * 0.00000123;
        x *= 1.000000001;
        if (x > 1e10) x *= 0.000000001;
    }
    /* use x so optimizer can't remove loop (no-op print) */
    if (x < 0) printf("weird: %f\n", x);
}
```

**Vision of the Department**

*To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.*

**Mission of the Department**

*To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.*

GNU nano 5.6.1 proces

```c
/* I/O-bound workload: append some data to file */
int io_work_once(const char *filename, int id, long iter) {
    int fd = open(filename, O_WRONLY | O_APPEND | O_CREAT, 0644);
    if (fd < 0) {
        fprintf(stderr, "open(%s) failed: %s\n", filename, strerror(errno));
        return -1;
    }
    char buf[128];
    int len = snprintf(buf, sizeof(buf), "worker %d iteration %ld\n", id, iter);
    ssize_t w = write(fd, buf, (size_t)len);
    if (w < 0) {
        fprintf(stderr, "write failed: %s\n", strerror(errno));
        close(fd);
        return -1;
    }
    close(fd);
    return 0;
}

void io_work(const char *filename, int id, long iterations) {
    for (long i = 0; i < iterations; ++i) {
        if (io_work_once(filename, id, i) != 0) {
            /* error already printed, but continue */
        }
    }
}

/* Worker wrapper for pthreads */
void *thread_worker(void *arg) {
    worker_arg_t *w = (worker_arg_t *)arg;
    if (!w) return NULL;
    if (w->filename == NULL) {
        /* CPU work */
        cpu_work(w->iterations);
    } else {
```

GNU nano 5.6.1 process_thread_perf.c

```c
    }
    return NULL;
}

/* Run using threads */
double run_threads(const char *mode, int workers, long iterations, const char *filename) {
    pthread_t *tids = calloc(workers, sizeof(pthread_t));
    worker_arg_t *args = calloc(workers, sizeof(worker_arg_t));
    struct timespec t0, t1;
    if (!tids || !args) {
        fprintf(stderr, "allocation failed\n");
        exit(1);
    }

    clock_gettime(CLOCK_MONOTONIC, &t0);
    for (int i = 0; i < workers; ++i) {
        args[i].iterations = iterations;
        args[i].filename = filename;
        args[i].id = i;
        if (pthread_create(&tids[i], NULL, thread_worker, &args[i]) != 0) {
            fprintf(stderr, "pthread_create failed\n");
            exit(1);
        }
    }

    for (int i = 0; i < workers; ++i) {
        pthread_join(tids[i], NULL);
    }
    clock_gettime(CLOCK_MONOTONIC, &t1);

    double elapsed = timespec_to_sec(&t1) - timespec_to_sec(&t0);
    free(tids);
    free(args);
    return elapsed;
```

**Vision of the Department**

*To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.*

**Mission of the Department**

*To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.*

GNU nano 5.6.1 process_thread_perf.c

```c
/* Run using processes (fork) */
double run_processes(const char *mode, int workers, long iterations, const char *filename) {
    pid_t *pids = calloc(workers, sizeof(pid_t));
    struct timespec t0, t1;
    if (!pids) {
        fprintf(stderr, "allocation failed\n");
        exit(1);
    }
    clock_gettime(CLOCK_MONOTONIC, &t0);
    for (int i = 0; i < workers; ++i) {
        pid_t pid = fork();
        if (pid < 0) {
            fprintf(stderr, "fork failed\n");
            exit(1);
        } else if (pid == 0) {
            /* child */
            if (filename == NULL) {
                cpu_work(iterations);
            } else {
                io_work(filename, i, iterations);
            }
            _exit(0);
        } else {
            pids[i] = pid;
        }
    }

    /* parent waits for all children */
    int status;
    for (int i = 0; i < workers; ++i) {
        waitpid(pids[i], &status, 0);
    }
    clock_gettime(CLOCK_MONOTONIC, &t1);
    free(pids);
    return timespec_to_sec(&t1) - timespec_to_sec(&t0);
```

GNU nano 5.6.1 process_thread_perf.c

```c
            fprintf(stderr, "io mode requires filename argument\n");
            return 1;
        }
        filename = argv[5];
    }

    if (workers <= 0 || iterations <= 0) {
        fprintf(stderr, "workers and iterations must be positive.\n");
        return 1;
    }

    printf("Mode: %s | Implementation: %s | workers=%d iterations=%ld %s\n",
           mode, use_threads ? "threads" : "processes", workers, iterations,
           (filename ? filename : ""));

    /* If IO mode and file exists, optionally truncate it to start fresh */
    if (filename) {
        int fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
        if (fd >= 0) close(fd);
        else fprintf(stderr, "warning: cannot truncate %s: %s\n", filename, strerror(errno));
    }

    double elapsed = 0.0;
    if (use_threads) {
        elapsed = run_threads(mode, workers, iterations, filename);
    } else {
        elapsed = run_processes(mode, workers, iterations, filename);
    }

    printf("Total elapsed time: %.6f seconds\n", elapsed);
    printf("Average time per worker: %.6f seconds\n", elapsed / workers);
    return 0;
```

GNU nano 5.6.1

```
CC = gcc
CFLAGS = -O2 -Wall -Wextra -pthread
TARGET = process_thread_perf
SRC = process_thread_perf.c

all: $(TARGET)

$(TARGET): $(SRC)
	$(CC) $(CFLAGS) -o $(TARGET) $(SRC)

clean:
	rm -f $(TARGET)
```

| | |
|---|---|
| | Output:<br>`[lab1@localhost process_thread_project]$ ./process_thread_perf -t cpu 4 20000000`<br>`Mode: cpu \| Implementation: threads \| workers=4 iterations=20000000`<br>`Total elapsed time: 0.101108 seconds`<br>`Average time per worker: 0.025277 seconds`<br>`[lab1@localhost process_thread_project]$ ./process_thread_perf -p cpu 4 20000000`<br>`Mode: cpu \| Implementation: processes \| workers=4 iterations=20000000`<br>`Total elapsed time: 0.101084 seconds`<br>`Average time per worker: 0.025271 seconds`<br>`[lab1@localhost process_thread_project]$ ./process_thread_perf -p io 4 5000 /tmp/testio.txt`<br>`Mode: io \| Implementation: processes \| workers=4 iterations=5000 /tmp/testio.txt`<br>`Total elapsed time: 0.026515 seconds`<br>`Average time per worker: 0.006629 seconds`<br>`[lab1@localhost process_thread_project]$` |
| Output Analysis | The experimental results typically show that threads outperform processes in terms of execution time and resource utilization for I/O-bound or shared-memory tasks, since threads communicate faster within the same address space. However, for CPU-intensive tasks on multi-core systems, processes may perform better due to true parallelism without the Global Interpreter Lock (in languages like Python). Hence, the performance gain depends on the task type, system architecture, and implementation efficiency. |
| Github Link | https://github.com/Prerana-Bijekar/HPC |
| Conclusion | The comparison between processes and threads reveals that while threads offer lower overhead and faster execution for concurrent operations, processes provide better isolation and fault tolerance. Threads are ideal for lightweight, shared-memory applications, whereas processes are better suited for computationally intensive or independent tasks. Ultimately, choosing between them depends on the balance between performance, reliability, and resource management needs. |
| Plag Report (Similarity index < 12%) |  |
| Date | 1 November 2025 |