POSIX Process and Thread Management Notes

1. `fork()` and Process Creation

- `fork()` creates a new process (child) by duplicating the calling process.
- Return values:
  - `0` in the child process.
  - Child PID in the parent process.
  - `-1` if the fork fails.
- The child process receives a copy-on-write duplicate of the parent's memory and file descriptor table.
- Use `wait()` or `waitpid()` in the parent to reap the child and avoid zombie processes.
- Each child should call `exit()` (or return from `main()`) to terminate cleanly.

2. `exec()` Family of Functions

All `exec*()` functions replace the current process image with a new program.

Variants

- `execl(path, arg0, ..., NULL)` — List of arguments; full path required.
- `execlp(file, arg0, ..., NULL)` — List + `PATH` lookup.
- `execle(path, arg0, ..., NULL, envp)` — List + custom environment.
- `execv(path, argv[])` — Vector of arguments; full path.
- `execvp(file, argv[])` — Vector + `PATH` lookup.
- `execvpe(file, argv[], envp[])` — Vector + `PATH` + custom environment (GNU-only).

Usage Table

Function	Use When	PATH Lookup
`execl`	Static args, full path	No
`execlp`	Static args, uses `PATH`	Yes
`execle`	Static args, custom env	No
`execv`	Dynamic args, full path	No
`execvp`	Dynamic args, uses `PATH`	Yes
`execvpe`	Dynamic args, `PATH`, env	Yes (GNU-only)

3. POSIX Threads (`pthread`)

- Java-style concurrency uses threads; POSIX provides `pthread` for this.
- `pthread_create()` spawns a thread that shares the same address space.
- `pthread_join()` waits for a thread to finish.
- Threads can lead to race conditions and require proper synchronization.
- Threads are more lightweight than processes but less isolated.

4. Semaphores

- Semaphores (`sem_t`) are integer counters used to control access to shared resources.
- `sem_init(sem, pshared, value)` — Initialize semaphore to given value.
- `sem_wait(sem)` — Decrement; blocks if value is 0.
- `sem_post(sem)` — Increment; unblocks one waiter if any.
- `sem_destroy(sem)` — Cleans up; does not free memory.
- Used for mutual exclusion (binary semaphores) or limiting access (counting semaphores).

Dining Philosophers: Limit-Seat Strategy

- Using a semaphore initialized to $N - 1$ prevents deadlock in the dining philosopher problem.
- Limits the number of philosophers who can attempt to pick up chopsticks to ensure progress.
- Prevents circular wait, breaking one of Coffman's deadlock conditions.
- Starvation is still possible due to unfair scheduling.

Key Terms

- **Turnaround time**: Total time from job arrival to completion.
- **Response time**: Time from job arrival to first CPU execution.
- **Waiting time**: Time a job spends in the ready queue.
- **Throughput**: Number of jobs completed per unit time.

1. First-Come First-Served (FCFS)

- **Policy**: Run processes in the order they arrive.
- **Preemptive?** No
- **Response Time**: Can be poor for short jobs after long ones.
- **Turnaround Time**: High if long jobs precede short ones.
- **Advantages**: Simple, fair in arrival order.
- **Disadvantages**: Convoy effect; poor average response time.

2. Shortest Job First (SJF)

- **Policy**: Run the job with the shortest total CPU burst time.
- **Preemptive?** No
- **Response Time**: Good for short jobs.
- **Turnaround Time**: Optimal (provably minimal) if job length is known.
- **Advantages**: Minimizes average turnaround time.
- **Disadvantages**: Requires knowing job length; may cause starvation.

3. Shortest Remaining Time (SRT)

- **Policy**: Preemptive version of SJF; always run job with shortest remaining time.
- **Preemptive?** Yes
- **Response Time**: Very good for short jobs.
- **Turnaround Time**: Optimal under perfect prediction.
- **Advantages**: Great for interactive tasks.
- **Disadvantages**: Starvation of long jobs; requires accurate prediction.

4. Round Robin (RR)

- **Policy**: Fixed time quantum; cycle through ready queue.
- **Preemptive?** Yes
- **Response Time**: Generally good, especially for short jobs.
- **Turnaround Time**: Depends on quantum size.
- **Advantages**: Fair, responsive for time-shared systems.
- **Disadvantages**: Poor performance if quantum is too small or too large; context switch overhead.

5. Lottery Scheduling

- **Policy**: Processes hold "lottery tickets"; winner gets CPU.
- **Preemptive?** Yes (depends on implementation)
- **Response Time**: Fair on average.
- **Turnaround Time**: Probabilistically fair.
- **Advantages**: Probabilistic fairness, flexible priority control.
- **Disadvantages**: Randomized; no strict guarantees.

6. Multi-Level Feedback Queue (MLFQ)

- **Policy**: Multiple queues with increasing priorities; new jobs start at high priority and move down if they consume more CPU.
- **Preemptive?** Yes
- **Response Time**: Excellent for interactive jobs.
- **Turnaround Time**: Adaptive; favors short and I/O-bound jobs.
- **Advantages**: Dynamic, responsive, balances fairness and performance.
- **Disadvantages**: Complex to tune; starvation possible without aging.

Comparison Table

Algorithm	Preemptive	Fairness	Response Time	Turnaround Time
FCFS	No	Arrival-order fair	Poor	High (convoy effect)
SJF	No	No	Excellent for short jobs	Optimal (theoretical)
SRT	Yes	No	Best for short jobs	Optimal
RR	Yes	Yes	Good	Medium (depends on quantur
Lottery	Optional	Probabilistic	Fair on average	Fair on average
MLFQ	Yes	Adaptive	Excellent	Adaptive