DSC204A: Scaleable Data Systems Winter 2025

8: Process

Instructor: Umesh Bellur *Scribe: Marcus Chang*

1 Process API - Interfacing with the OS

1.1 API

- CREATE - fork(): Creates an identical copy of the process as a child
- WAIT - wait(): Waits for a process to complete execution
- EXECUTE - execvp*(): Starts a program programmatically!
- STOP - kill(): Sends a signal to a process

1.2 Example

Consider the following code:

```
import os

os.fork()
os.fork()
print("Hi")
```

Each `fork()` duplicates the current process. So after two forks, we have $2^2 = 4$ processes.

Explanation:

- First `fork()` splits P0 into P0 and P1.
- Second `fork()` is called by both P0 and P1, doubling each again:
  - P0 creates P2
  - P1 creates P3
- Each of the 4 processes independently executes `print("Hi")`.

Total output: 4 lines of `Hi`

2 Multiprocessing

2.1 Requirements on process execution

- **Speed**: Minimize overhead in executing user programs
  - **Direct Execution**: Let user programs run directly on CPU without an intermediary. (exist issues)
  - Solution:
    - **Limited Direct Execution**: User programs run directly but with restrictions.
- **Fairness**: Ensure all processes get a fair share of CPU time.
  - Solution:
    - **Context Switching**: Save and restore process states to switch between them.
    - **Timer Interrupts**: Periodically interrupt running processes to enforce time-sharing.
- **Isolation**: Prevent Processes from affecting each other's execution or memory.
  - Solution:
    - **Memory Isolation**: Each process operates in its own protected memory space.

2.2 Scheduling Policies/Algorithms

2.2.1 Concepts

- **Schedule**: Record of what process runs on each GPU & when
- **Policy**: Control how OS time-shares CPUs among processes
- **key terms**
  - Arrival Time: Time when process gets created
  - Job Length: Duration of time needed for process
  - Start Time: Times when process first starts on processor
  - Completion Time: Time when process finishes/killed
  - Response Time: Start Time - Arrival Time
  - Turnaround Time: Completion Time - Arrival Time
- **Preemption**: In general, OS does not know all Arrival Times and Job Lengths beforehand, but CPU allow cut line.
- **The trade-off in scheduling between overall worklead performance and allocation faitness**

2.2.2 Context Switch Mechanism

The figure1 illustrates the **Context Switch Mechanism**, which is essential for ensuring **fairness** in CPU scheduling by allowing the operating system to switch between processes. The step-by step explanation of the figure1:

1. P1 is executing
2. An interrupt or system call occurs
3. Save P1's state to **Process Control Block 1** (PCB1)
4. Load P2's state from PCB2
5. P2 start executing
6. Another interrupt or system call
7. Save P2's state
8. Reload P1's state
9. P1 resumes execution

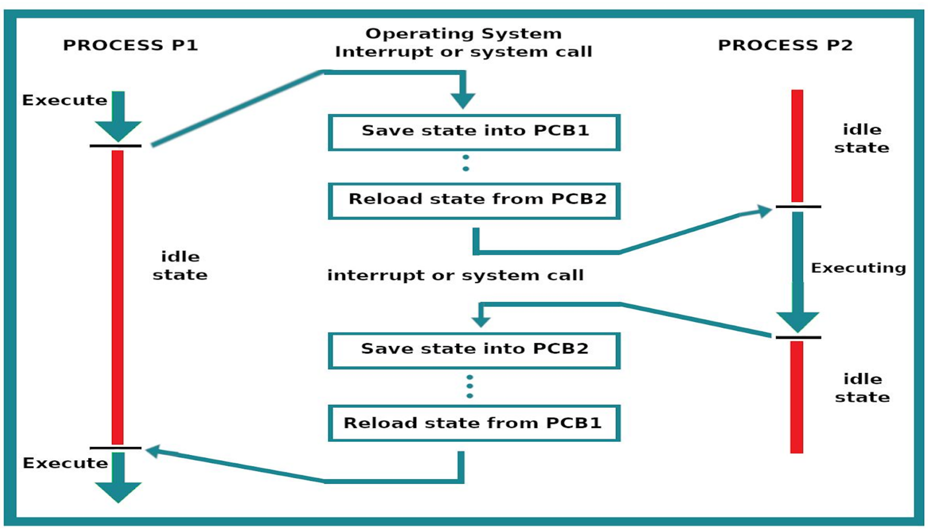

Figure 1: Context Switch Mechanism

Memory Components Involved in Context Switching

- **Process Control Block (PCB)**: Each process has its own PCB, which resides in kernel memory.
  - The OS saves the current process's into its PCB and loads the next process's state from PCB.
  - It stores CPU register, memory manage info, scheduling info, process state.
- **Page Tables / Virtual Memory**: Each process has its own virtual address space and page table.

- The OS updates the page table base register (PTBR) or MMU context to point to the new process's memory mapping.
- This ensures memory isolation, so one process can't access another's memory.

2.2.3 Scheduling Policies

- FIFO (First-In-First-Out): Run processes in the order they arrive
  - Pros: Simple and fair in terms of arrival time
  - Cons: Long jobs can block short ones
  - Use Case: Batch processing with predictable workloads
- SJF (Shortest Job First): Run the job with the shortest total execution time first
  - Pros: Minimizes average waiting time
  - Cons: Requires knowing job lengths in advance
  - Use Case: Ideal in controlled environments like simulations.
- SCTF (Shortest Completion Time First): Preempt current job if a newly arrived job has a shorter remaining time
  - Pros: Even better average response time than SJF
  - Cons: Preemption adds overhead; long job may be delayed indefinitely
  - Use Case: Real-time or interactive system with know burst times
- Round Robin: Give each process a fixed time slice in a rotating order
  - Pros: Ensures fairness; no process waits too long
  - Cons: Context switching overhead; poor performance if time slice is too small or too big
  - Use Case: General-purpose time-sharing system