OPERATING SYSTEMS

by
Marwa Yusuf

Lecture 10
Thursday 26-11-2020

Chapter 2 (2.4.3)

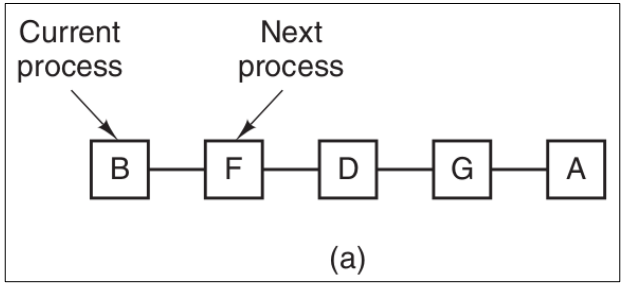
Processes and Threads

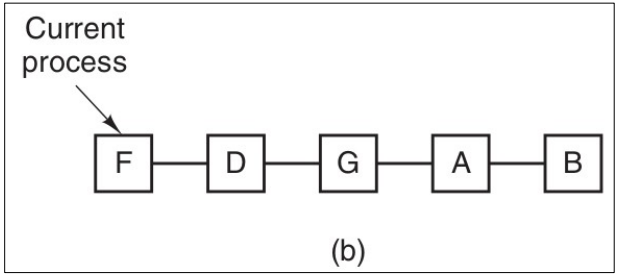
Scheduling in Interactive Systems

- Remember:
  - General Requirements:
    - Fairness.
    - Policy enforcement.
    - Balance.
  - Specific Requirements:
    - Response time.
    - Proportionality.

Round-Robin Scheduling

- One of the oldest, simplest, fairest, and most widely used.
- Preemptive.
- Each process is preempted after a **quantum** time of execution.
- If the processes finishes before end of quantum → switching.
- The choice of quantum length vs switching time,
  - ex: 4 msec quantum with 1 msec switching time → 20% waste
  - Quantum = 100 msec → 1% waste
  - Ex: 50 server requests, the last one may wait for 5 sec (unacceptable)
  - Quantum > mean execution burst time → most switches are not preempting.
- Conclusion:
  - Too short quantum → too much switches → lower CPU efficiency.
  - Too long quantum → poor responsiveness, specially with short interactive requests.
  - 20-50 msec is usually a reasonable choice.



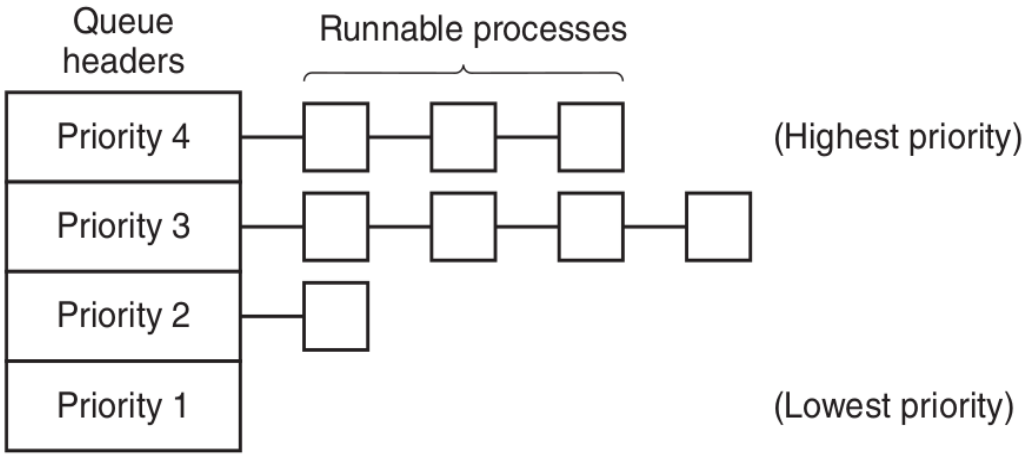


Priority Scheduling

- Round robin assumes that all processes are equally important. What about VIPs? (users or processes)
  - President process vs student process.
  - Email sending process vs video displaying process.
- The process with the highest assigned priority is run first.
- To prevent high priority processes from monopolizing CPU:
  - Lower priority periodically (ex: at each clock tick)
  - Assign a quantum, after which the next high priority process is run.
- Priorities are assigned statically:
  - General vs captain process, 100$ vs 70$ process.
  - *Nice* command in UNIX: voluntarily reduce my priority. (who may use that?)
- Or dynamically:
  - I/O-bound processes get higher priorities. Ex: priority = 1/f where f = the fraction used of last quantum.

Priority Scheduling (cont.)

- Priority classes:
  - Each class has a priority.
  - Priority scheduling between classes, round-robin within each class.
  - Lower classes may starve.



Multiple Queues

- CTSS system:
  - One process in memory → very long switching time → sol: long quantum to CPU-bound processes once in a while → worse response time for others.
  - Priority classes where 1st class execute for 1 quantum, 2nd class for 2 quanta, … etc. Processes move down after each run.
  - Ex: 100 quanta process: 1, 2, 4, 8, 18, 32, 64(37) → only 7 switches instead of 100 → moving down in priority, more time for interactive short processes.
  - To avoid punishing forever, if a process gets a carriage return (enter key) stroke, it is moved to the highest priority (it is getting interactive).
    - What about that scheming user who discovered the trick?
    - What looks good does not necessarily run well.

Shortest Process Next

- Cycles of wait for command, execute command.
- Assume that execution of each command is a separate “job”.
- How to decide which is the shortest?
- One approach: estimate based on past behavior:
  - Estimated = T_0 , Measured = T_1
    $T_{new} = a\ T_0 + (1\text{-}a)\ T_1$
  - *A* determines if history is forgotten quickly or not.
  - If a=0.5 (which is easy to implement)→ T_0 , $T_0/2 + T_1/2$, $T_0/4 + T_1/4 + T_2/2$, $T_0/8 + T_1/8 + T_2/4 + T_3/2$
  - This technique sometimes called **aging**.

Guaranteed Scheduling

- Make a promise (to a user for example) and stick to it.
- Example promise: Each user (each process in a single user system) gets a fair share of the CPU time (1/n of CPU cycles).
  - Compute the ratio: (what the process actually got since creation) / (what it is entitles to: time since creation / n)
  - The process with lowest ratio is run till it reaches the next low ratio.

Lottery Scheduling

- Easier to implement than guaranteed scheduling.
- Each process holds a number of lottery tickets. The system periodically chooses a lottery number. The winner process gets the CPU. (ex: choose 50 times/second, giving 20 msec of CPU prize).
- The fraction of tickets a process holds affects the probability of winning. In the long run, it is almost accurate.
- Advantages:
  - Highly responsive: new coming processes can win soon.
  - Cooperating processes may exchange tickets. Ex: client and server processes.
  - Solve problems in some difficult situations. Ex: video streaming processes with different frame rates.

Fair-Share Scheduling

- Assume User 1 with 9 processes and User 2 with 1 process → User 1 gets 90% of CPU time.
- Take into account the owner of the process when scheduling.
- Ex: User 1 (A, B, C, D) & User 2 (E) with round robin
  - %50 to 50%: A E B E C E D E A E B E C E D E …
  - User 1 twice User 2: A B E C D E A B E C D E ...