CPU Scheduling (cont)

ECE595
Sep 18

Y. Charlie Hu

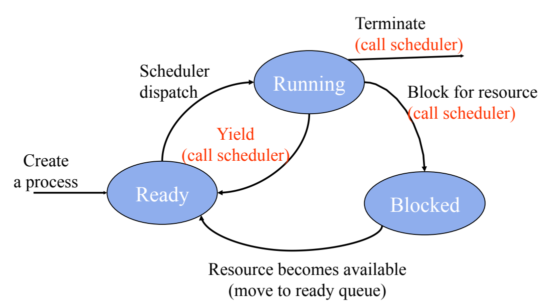
Midterm

- Oct 4, 2018: 7-9pm
- Location: TBD

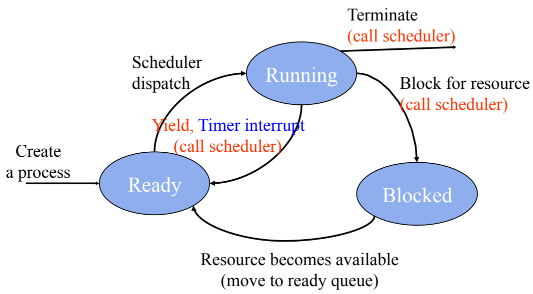
Timesharing Systems

- Timesharing systems support interactive use
  - each user feels he/she has the entire machine
    - CPU, memory, I/O devices
- CPU - How?
  - optimize response time
  - based on time-slicing

Process State Transition of Non-Preemptive Scheduling



Preemptive Scheduling



Review on CPU scheduling

- Mechanism is easy, policy is hard
  - Jobs have diverse characteristics
  - 4 performance metrics
  - Don't know about future
- Hard to analyze even when narrowing down metric/job nature
  - FIFO vs. Round Robin in average turnaround time
    - 10 procs, 100 sec each, who wins?
    - 10 procs, 100 sec for first, 10 sec other 9, who wins?
  - Adding I/O to the diversity
- STCF vs. SRTCF

STCF vs. SRTCF

- Shortest time to completion first (shortest job first)
  - Non-preemptive
- Shortest *remaining* time to completion first
  - Preemptive
- When (what job arrival secenario) are they the same?
- Advantage
  - Minimal average turnaround time
- Disadvantage
  - Can cause starvation; difficult to know the future

Deep thinking

- Can you do better than Shortest job first in terms of average turnaround time?
- Prove it!

Real challenge in using STCF

- Used in long-term scheduling batch system
  - e.g., explicit-length job queues (for repeated jobs)
- What about short-term scheduling (e.g. time-sharing system)?
  - Try prediction for repeated jobs
  - Exponential average:
    Pred(n+1) = alpha*t_n + (1-alpha)*Pred(n)
- What about starvation?
  - Aging

Observations so far

- Need to accommodate interactive jobs
  - Need some kind of RR
- Diversity in jobs – job length, I/O mix
  - RR also appears to help
- SJF also has virtue
  - Reduce avg. turnaround time
- Can we accommodate all?

Scheduling policies

FIFO	Response time
	Throughput
RR	
	Avg. turnaround time
SJF	
	Fairness

Break

True or false?

- “A CPU scheduling algorithm that minimizes avg. turnaround time cannot lead to starvation.”
- “Among all CPU scheduling algorithms, Round Robin always gives the worse average turnaround time.”

Priority Scheduling

- To accommodate the spirits of SJF/RR/FIFO
- The method
  - Assign each process a *priority*
  - Run the process with highest priority in ready queue first
    - Use FIFO for processes with equal priority
  - Adjust priority dynamically
    - To deal with *all* issues: e.g. aging, I/O wait raises priority
- Advantage
  - Flexibility: Not all processes are "born" equal
- Challenge?

Priority Scheduling (cont)

- Who sets the priorities?
  - Internally by OS
    - I/O to computation ratio (can be dynamic)
    - Memory requirement (can be dynamic)
    - Time constraints (e.g. real-time systems)
  - Externally by users/sysadm
    - Importance
    - Funds paid for
    - e.g. nice
- How -- Dynamically adjustment is tricky

Approach 1: Multiple Queue Scheduling

- Motivation: processes may be of different nature and can be easily classified
  - e.g. foreground jobs vs. background jobs
- The method:
  - Processes permanently assigned to one queue, based on processes priority / type
  - Each queue has its own scheduling algo.
    - e.g. RR for foreground queue, FCFS for background queue
  - Need a scheduling among the queues
    - e.g. fixed priority preemptive scheduling (high-pri queue trumps other)
    - e.g. time-slice between queues

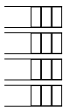
Pros/Cons of Multiple Queue Scheduling

- Pros:
  - Low scheduling overhead
    - Jobs do not move across queues
- Cons:
  - Processes permanently assigned to one queue – not flexible
    - Program behavior may change
    - E.g. can switch between I/O bound and CPU bound

    → Need some learning/adaptation at runtime
  - Starvation cannot be easily handled

    → Need some learning/adaptation at runtime

Approach 2: Multilevel Feedback Queue



Priority	Time slices
0	1
1	2
2	4
3	8

- Jobs start at high priority queue
- Feedback
  - If timeout expires, drop one level
  - If waiting too long, push up one level (aging)
- Leave I/O bound and interactive processes in higher-priority queue

UNIX System V

- One of the first commercial versions of the Unix
- Originally developed by AT&T
- First released in 1983
- 4 major versions, termed R1 R2, R3, and R4
- SVR4, commercially most successful version, the result of Unix System Unification (with collaborations of major UNIX vendors)
  - IBM's AIX is based on SVR3
  - Sun 's Solaris and HP's HP-UX are based on SVR4
- 80s-early 90s, System V and BSD were the two major "flavors" of UNIX
  - Large multi-user systems vs. Desktop workstations

Traditional Unix Scheduling (SVR3, 4.3 BSD)

Multilevel Feedback Queue with 1 sec preemption

- "1 sec" preemption
  - Preempt if a process doesn't block or complete within 1 sec
- Priority is recomputed every sec (low # is higher)

  P_i = base + CPU_{i-1}/ 2 + nice, where $CPU_i = (U_i + CPU_{i-1}) / 2$

  base is the base priority of the process

  U_i is process utilization in interval i, nice in [-20,20]
- Base priorities
  - Virtual memory Swapper
  - Block I/O device control
  - Character I/O device control
  - User processes

Scheduling Algorithms in OSes

Operating System	Preemption	Algorithm
Windows 3.1x	None	Cooperative Scheduler
Windows 95, 98, Me	Half	Preemptive for 32-bit processes, Cooperative Scheduler for 16-bit processes
Windows NT (2000, XP, Vista, 7, and Server)	Yes	Multilevel feedback queue
Mac OS pre-9	None	Cooperative Scheduler
Mac OS 9	Some	Preemptive for MP tasks, Cooperative Scheduler for processes and threads
Mac OS X	Yes	Multilevel feedback queue
Linux pre-2.6	Yes	Multilevel feedback queue
Linux 2.6-2.6.23	Yes	O(1) scheduler
Linux post-2.6.23	Yes	Completely Fair Scheduler
Solaris	Yes	Multilevel feedback queue
NetBSD	Yes	Multilevel feedback queue
FreeBSD	Yes	Multilevel feedback queue

Cooperative Scheduling

- Early multitasking systems used applications that voluntarily ceded time to one another.
- This approach, which was eventually supported by many OSes, is known today as cooperative multitasking.
- Cooperative multitasking was once the scheduling scheme employed by Microsoft Windows (prior to Windows 95 and Windows NT) and Mac OS (prior to Mac OS X) to enable multiple applications to be run simultaneously
- Now rarely used in larger systems

Example Details in Chapter 5

- Solaris 2
- Windows XP
- Linux

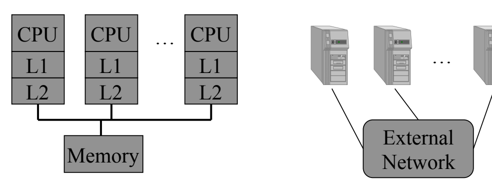
- All are variations of Multilevel Feedback Queue

Mechanism vs policy: Lottery Scheduling

- Motivation
  - Priority scheduling is implemented by adjusting priorities
  - Adjusting priority is a bit ad hoc (e.g. what rate?)

- Lottery method [OSDI 94]
  - Give each job a number of tickets
  - High priority jobs get more tickets
  - Randomly pick a winning ticket
  - To avoid starvation, give each job at least one ticket

Multiprocessor and Cluster



Multiprocessor architecture
- L2 cache coherence
- A single "image" OS

Cluster/Multicomputer
- Distributed memory
- An OS on each box

Multiprocessor/Cluster Scheduling

- New design issue: process/thread inter-dependence
  - Threads of the same process may synchronize
  - Processes of the same job may send/recv messages

Multiprocessor/Cluster Scheduling: Example Approach

- Gang scheduling (coscheduling)
  - Threads of same process will run together on multiprocessor
  - Processes of same application run together on cluster

Reading assignment

- OSC, Chapter 5