Scheduling: Lottery Scheduling

Operating System (OS)

안인규

강의자료는 KAIST 원유집 교수님 OS 강의 자료를 참고했습니다.

Proportional Share Scheduler

- Fair-share scheduler
  - Guarantee that each job obtain *a certain percentage* of CPU time.
  - Not optimized for turnaround or response time

Basic Concept

- Tickets
  - Represent the share of a resource that a process should receive
  - The percent of tickets represents its share of the system resource in question.

- Example
  - There are two processes, A and B.
    - Process A has 75 tickets → receive 75% of the CPU
    - Process B has 25 tickets → receive 25% of the CPU

Lottery scheduling

- The scheduler picks a winning ticket.
  - Load the state of that *winning process* and runs it.
- Example
  - There are 100 tickets
    - Process A has 75 tickets: 0 ~ 74
    - Process B has 25 tickets: 75 ~ 99

Scheduler's winning tickets:	63	85	70	39	76	17	29	41	36	39	10	99	68	83	63
Resulting scheduler:	A	B	A	A	B	A	A	A	A	A	A	B	A	B	A

두 작업이 경쟁하는 시간이 길어질수록,
원하는 비율을 달성할 가능성이 높아짐

Ticket Mechanisms

- Ticket currency
  - A user allocates tickets among their own jobs in whatever currency they would like.
  - The system converts the currency into the correct global value.
  - Example
    - There are 200 tickets (Global currency)
    - Process A has 100 tickets
    - Process B has 100 tickets

User A → *500* (A's currency) to A1 → *50* (global currency)
→ *500* (A's currency) to A2 → *50* (global currency)

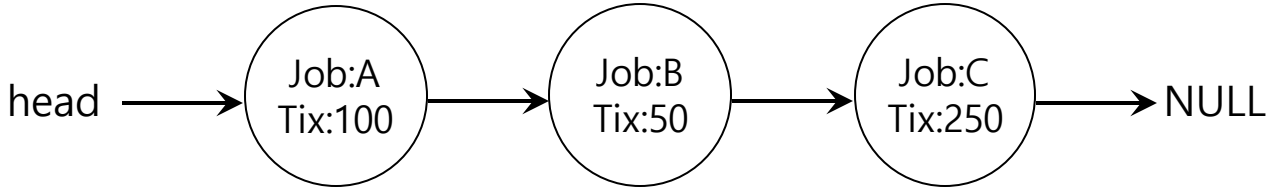
User B → *10* (B's currency) to B1 → *100* (global currency)

Ticket Mechanisms (Cont.)

- Ticket transfer
  - 프로세스는 일시적으로 자신의 티켓을 다른 프로세스에 양도할 수 있음
- Ticket inflation
  - 프로세스가 일시적으로 소유한 티켓 수를 증가하거나 감소시킬 수 있음
  - 특정 프로세스가 더 많은 CPU 시간이 필요하면, 티켓을 증가시켜 할당량을 늘릴 수 있음

Implementation

- Example: There are there processes, A, B, and C.
  - Keep the processes in a list sorted with the ticket size: highest ticket first



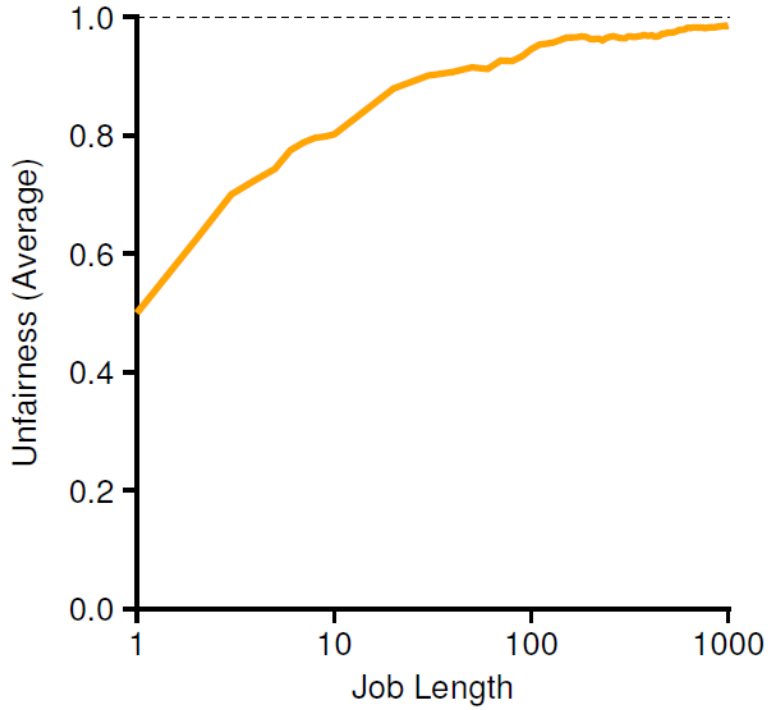

```
1         // counter: used to track if we've found the winner yet
2         int counter = 0;
3
4         // winner: use some call to a random number generator to
5         // get a value, between 0 and the total # of tickets
6         int winner = getrandom(0, totaltickets);
7
8         // current: use this to walk through the list of jobs
9         node_t *current = head;
10
11        // loop until the sum of ticket values is > the winner
12        while (current) {
13                  counter = counter + current->tickets;
14                  if (counter > winner)
15                            break; // found the winner
16                  current = current->next;
17        }
18        // 'current' is the winner: schedule it...
```

Implementation (Cont.)

- U: fairness metric
  - The time the first job completes divided by the time that the second job completes.
  - 모든 job이 동시에 시작했다고 가정
- Example:
  - There are two jobs, each jobs has runtime 10.
    - First job finishes at time 10
    - Second job finishes at time 20
  - $U = \frac{10}{20} = 0.5$
  - U will be close to 1 when both jobs finish at nearly the same time.

Lottery Fairness Study

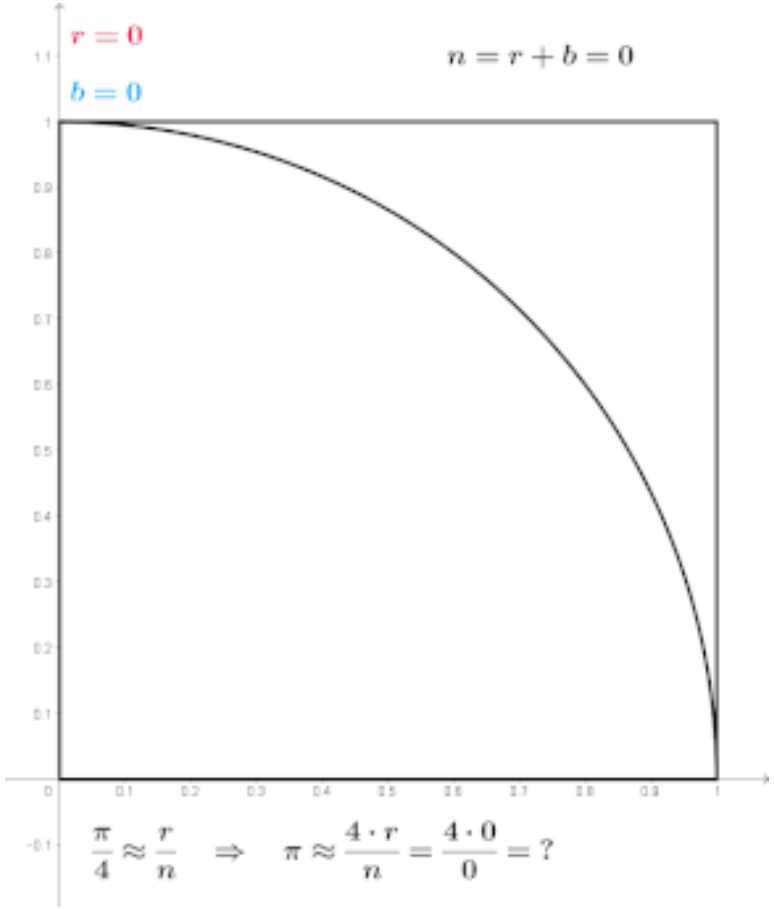
- There are two jobs.
  - Each jobs has the same number of tickets (100).



When the job length is <u>not very long</u>, average fairness can be quite severe.

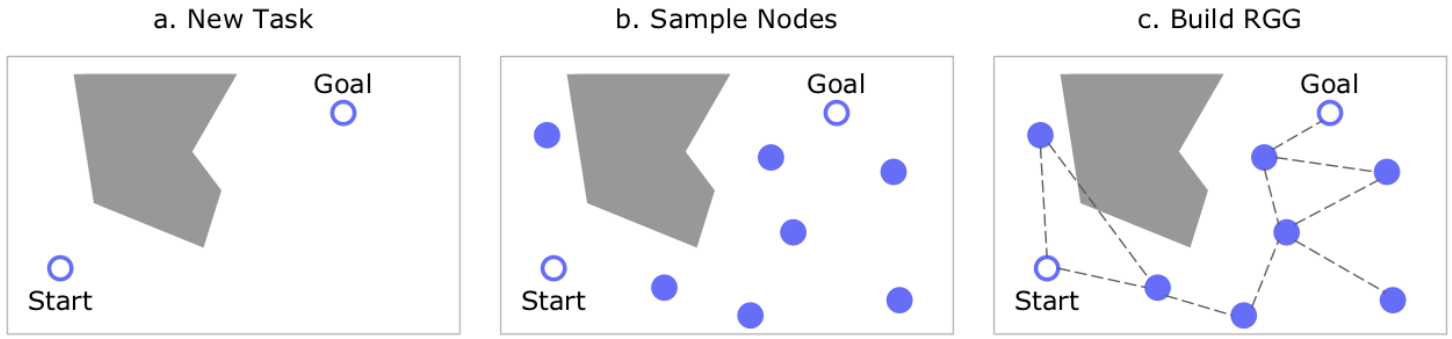
Extra: Monte Carlo Sampling

- Monte Carlo Sampling?
  - repeated random sampling to obtain numerical results

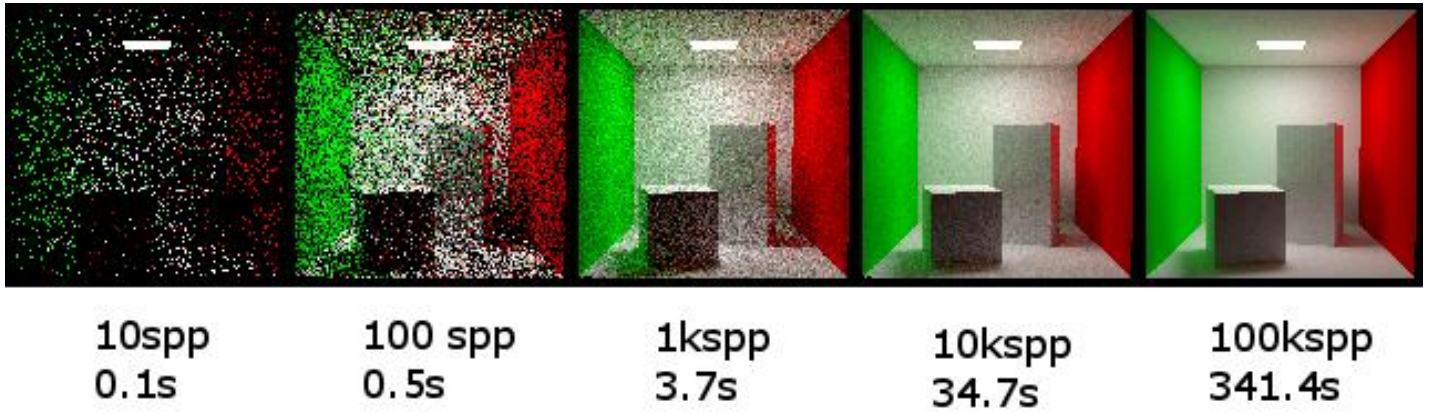


Extra: Monte Carlo Sampling

- Monte Carlo Sampling? - 어디서 사용되나?



Sampling-based Path Planning (경로생성)



Monte Carlo Ray Tracing (렌더링)

Deterministic Approach: Stride Scheduling

- Stride of each process
  - (A large number) / (the number of tickets of the process)
  - Example: A large number = 10,000
    - Process A has 100 tickets → stride of A is 100
    - Process B has 50 tickets → stride of B is 200

- A process runs, increment a counter(=pass value) for it by its stride.
  - Pick the process to run that has the lowest pass value

```
current = remove_min(queue);            // pick client with minimum pass
schedule(current);                      // use resource for quantum
current->pass += current->stride;       // compute next pass using stride
insert(queue, current);                 // put back into the queue
```

Stride Scheduling Example

Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	...

Stride Scheduling은 프로세스별 pass 값을 유지해야 함
새로운 프로세스는 pass 값이 0으로 설정 → CPU 독점!
Lottery Scheduling의 장점: 프로세스별 상태 정보를 유지할 필요가 없음

The Linux Completely Fair Scheduling (CFS)

- Completely Fair Scheduling (CFS)
  - The current CPU scheduler in Linux
  - Non-fixed timeslice.
    - CFS assigns process`s timeslice a proportion of the processor.
  - Priority (**Shortest job 기준이 아님**)
    - Enables control over priority by using nice value.
  - Efficient data structure.
    - Use red-black tree for efficient search, insertion and deletion of a process.

Basic

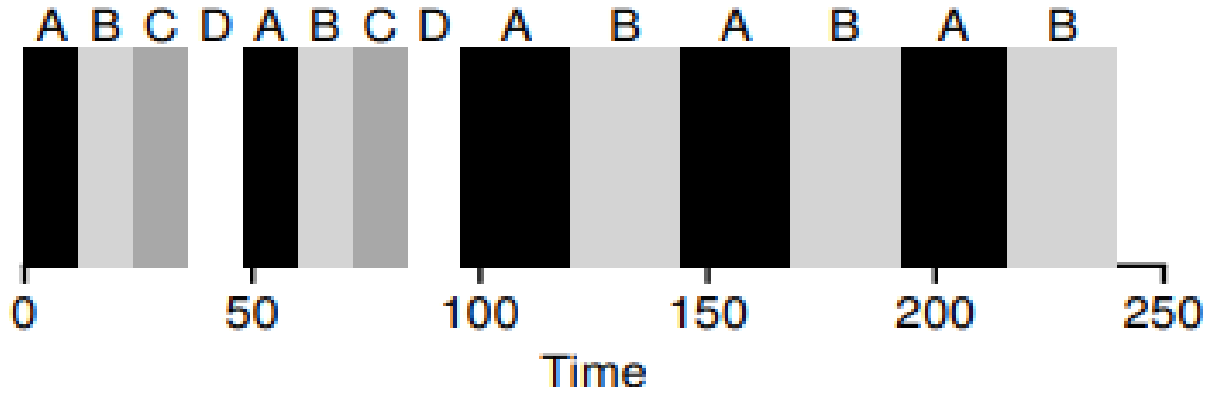
- Sched_latency
  - 모든 프로세스가 한 번씩 실행되는 목표 시간
  - A typical value is 48 (milliseconds)
  - Basic concept:

$$\text{Time Slice} = \frac{\text{sched\_latency}}{n} = \frac{48ms}{4} = 12ms$$

- Virtual runtime (vruntime)
  - Denote how long the process has been executing.
  - Per-process variable
  - CFS will pick the process with the **lowest vruntime** to run next.
  - 실행될 때의 물리적 시간 (physical time)에 비례하여 증가

Example

- Simple Example
  - 4 processes (A,B,C,D) and then 2 processes(C,D) complete.



- min_granularity
  - The minimum timeslice (6ms)
  - 너무 많은 프로세스가 실행 중일 때, 스케줄링 오버헤드에 너무 많은 시간이 소모되지 않도록 보장
  - If timeslice is 4.8ms?

Weight

- Nice value
  - CFS는 프로세스 우선순위 (priority)를 조정할 수 있는 기능을 제공
    - 사용자 또는 시스템 관리자가 특정 프로세스에 더 많은 CPU 사용 시간 할당 가능
  - Nice value: -20 (높은 우선순위) ~ +19 (낮은 우선순위)
  - 즉, nice value가 클수록 CPU를 덜 할당, 작을수록 더 할당
    - 너무 "nice" 하면 CPU를 양보하게 됨
  - The nice value is mapped to a weight (value is not important)

```
static const int prio_to_weight[40] = {
 /* -20 */     88761,     71755,     56483,     46273,     36291,
 /* -15 */     29154,     23254,     18705,     14949,     11916,
 /* -10 */      9548,      7620,      6100,      4904,      3906,
 /*  -5 */      3121,      2501,      1991,      1586,      1277,
 /*   0 */      1024,       820,       655,       526,       423,
 /*   5 */       335,       272,       215,       172,       137,
 /*  10 */       110,        87,        70,        56,        45,
 /*  15 */        36,        29,        23,        18,        15,
};
```

Timeslice with weight

- New timeslice formula

$$time\_slice_k = \frac{weight_k}{\sum_{n=0}^{n-1} weight_i} \cdot sched\_latency$$

- Simple Example
  - Assign Process `A` a nice value of -5 and process `B` a nice value of 0.

Process	nice value	weight	Time slice
A	-5	3121	36 ms
B	0	1024	12 ms

vruntime with weight

- vruntime formula
  - CFS는 vruntime도 우선순위에 맞게 조정해야 함
    - 높은 우선순위의 job이 더 긴 timeslice 할당 → runtime 증가 → 이를 보정해 줄 필요 있음
    - 왜? 높은 우선순위 job에게 CPU할당을 더 많이 하고 싶음
  - 실제 runtime을 계산 →프로세스 weight에 따라 반비례 조정

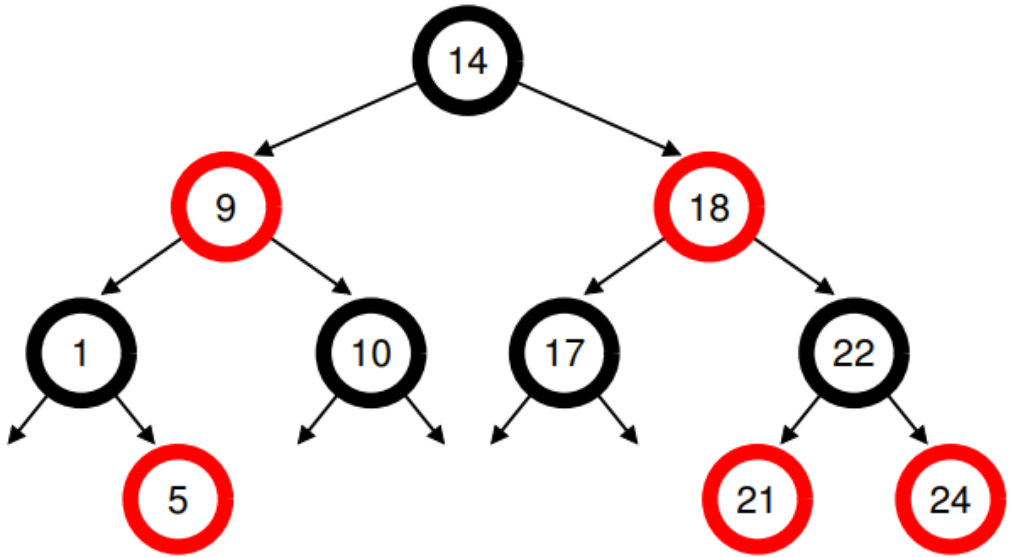
$$vruntime_i = vruntime_i + \frac{weight_0}{weight_i} \cdot runtime_i$$

  - Simple Example

Process	nice value	weight	Accumulated value
A	-5	3121	(1/3) * runtime
B	0	1024	1 * runtime

Structure of ready queue

- Red-Black Tree
  - Balanced binary tree (can address worst-case insertion)
  - Ordering of Red-Black Tree : **O(log n) (삽입, 삭제)**
  - Efficiently find the process with minimum virtual runtime
  - Only running (or runnable) processes are kept therein



IO and sleeping process

- Dealing with I/O and Sleeping processes
  - 일부 프로세스가 CPU를 독점하는 상황 방지 (starvation)
    - 프로세스가 IO request 후, vruntime이 매우 작아지는 경우 발생
      (B는 지속적으로 실행, A는 10초 동안 blocked state였다가 ready state)
  - 프로세스가 ready state가 될 때, 해당 프로세스의 vruntime을 tree에서 찾은 최소값보다 작지 않도록 설정
  - Jobs that sleep for short periods of time frequently do not ever get their fair share of the CPU