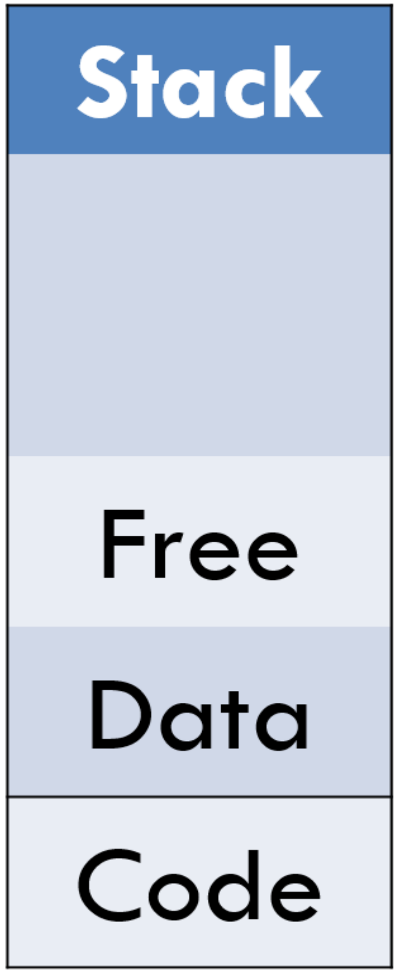
# AVL Tree

Application in Memory Management of Linux

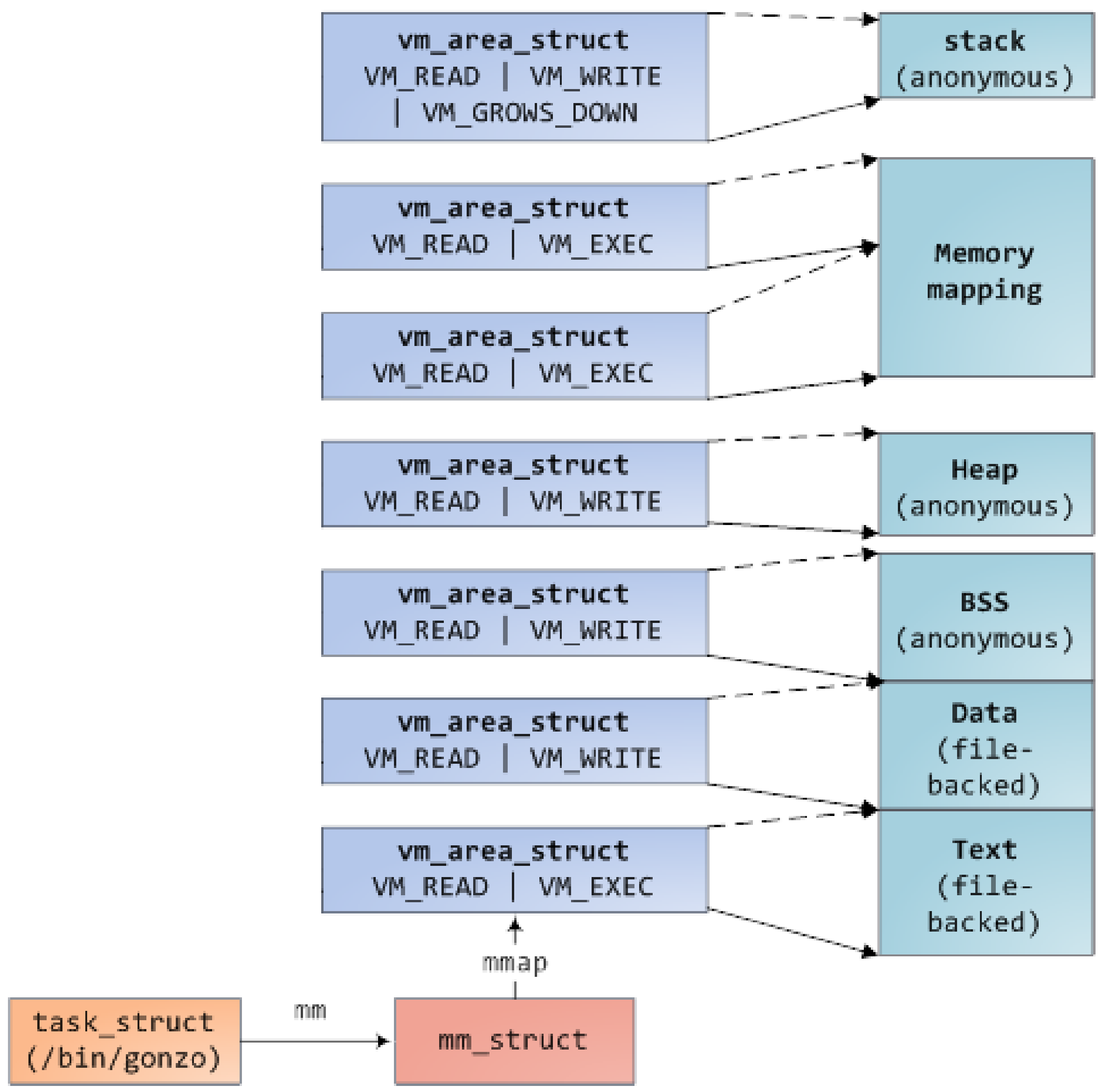
# Memory Management

- Virtual memory and Physical Memory
- Loading an executable image
- *Page Fault*



## Handling Page Fault

- Search the area in memory map to find which area is involved.
- Searching through the structures is critical to the efficient handling of page faults, these are linked together in an AVL tree



```
class mm_struct {
    //AVL Tree based ordered map
    class vm_area_struct* mmap_avl;
};
```

Based on "How the Kernel Manages Your Memory" by Gustavo Duarte 4

# Graphs

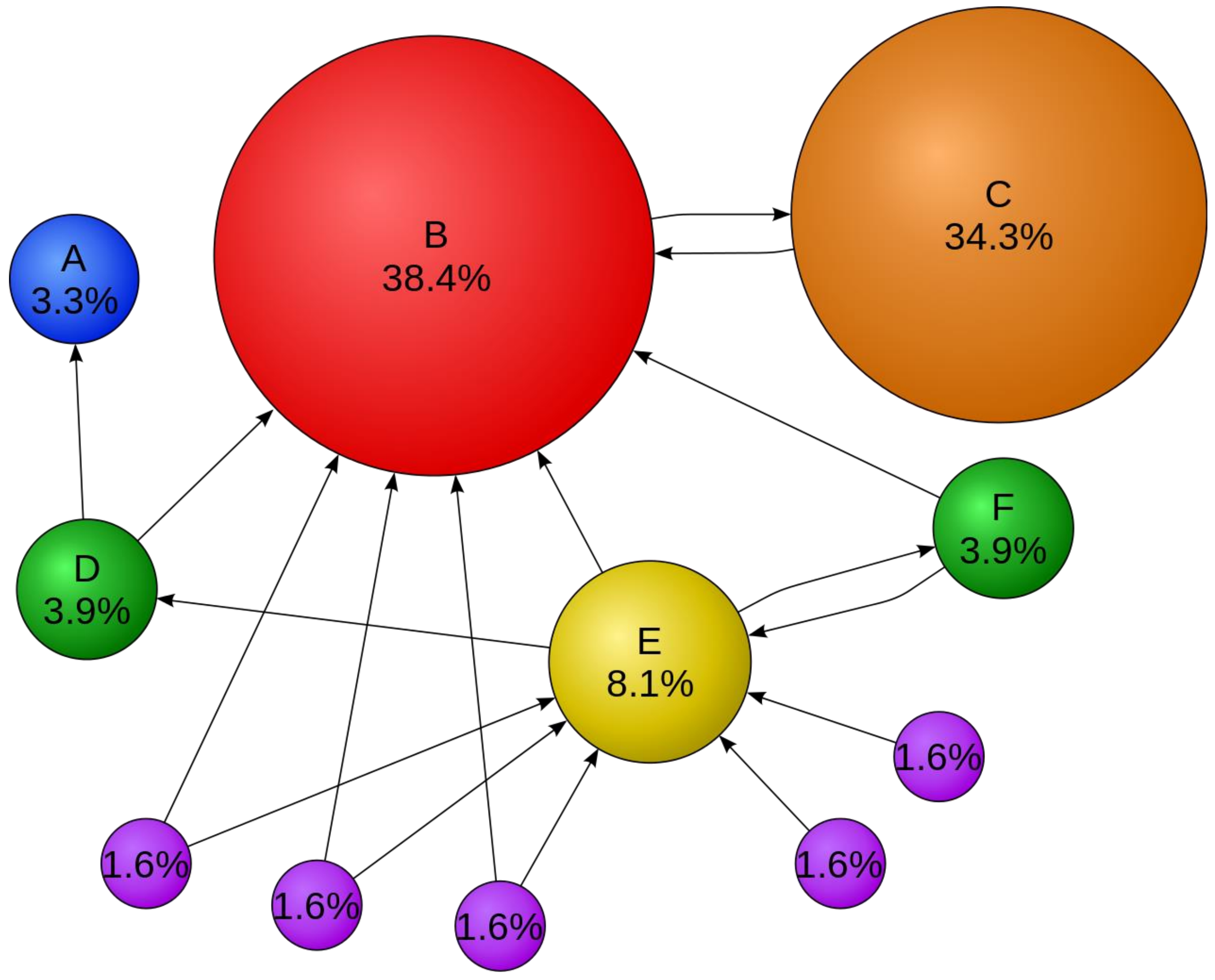
Application in Page Rank

## Page Rank

- Used to rank websites in search engine results.
- Counts the number and quality of links to a page
- Use directed Graph

$$PR(A) = \frac{1-d}{N} + d\left[\frac{PR(T1)}{C(T1)} + \frac{PR(T2)}{C(T2)} + \cdots + \frac{PR(Tn)}{C(Tn)}\right]$$

Based on Ian Rogers's "The Google Page rank Algorithm and How It Works"



d = 85%

Based on PageRank from Wikipedia

# Hash Table

Applications in dictionary clients

## Problem: Dictionary Lookup

- Receive a **C**omma-**S**eparated **V**alue (CSV) file.
- Sample Applications:
  - DNS Lookup
  - Amino Acids
  - Class list

```
TTT,Phe,F,Phenylalanine
TTC,Phe,F,Phenylalanine
TTA,Leu,L,Leucine
TTG,Leu,L,Leucine
TCT,Ser,S,Serine
TCC,Ser,S,Serine
TCA,Ser,S,Serine
…
```

amino.csv

```
TTT,Phe,F,Phenylalanine
TTC,Phe,F,Phenylalanine
TTA,Leu,L,Leucine
TTG,Leu,L,Leucine
TCT,Ser,S,Serine
TCC,Ser,S,Serine
TCA,Ser,S,Serine
TCG,Ser,S,Serine
TAT,Tyr,Y,Tyrosine
TAC,Tyr,Y,Tyrosine
TAA,Stop,Stop,Stop
TAG,Stop,Stop,Stop
TGT,Cys,C,Cysteine
TGC,Cys,C,Cysteine
TGA,Stop,Stop,Stop
TGG,Trp,W,Tryptophan
CTT,Leu,L,Leucine
CTC,Leu,L,Leucine
CTA,Leu,L,Leucine
CTG,Leu,L,Leucine
CCT,Pro,P,Proline
CCC,Pro,P,Proline
CCA,Pro,P,Proline
CCG,Pro,P,Proline
CAT,His,H,Histidine
CAC,His,H,Histidine
CAA,Gln,Q,Glutamine
CAG,Gln,Q,Glutamine
CGT,Arg,R,Arginine
CGC,Arg,R,Arginine
...
```

# Problem Details: Example Usage

Codon is key

Name is value

```
> csv_lookup amino.csv 0 3
ACT
Threonine
TAG
Stop
CAT
Histidine
```

# C++ Implementation

I/O

Creating Symbol Table

Processing Queries

```
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
#include <sstream>

using namespace std;

int main(int argc, char **argv) {
        ifstream in; in.open(argv[1]);
        int key_field = atoi(argv[2]);
        int val_field = atoi(argv[3]);

        map<string, string> mp;
        string line;
        if (in.is_open()) {
                while (getline(in, line, '\n')) {
                        stringstream ss(line);
                        string token, key_token, val_token;
                        int ind = 0;
                        while (getline(ss, token, ',')) {
                                if (ind == key_field) key_token = token;
                                else if (ind == val_field) val_token = token;
                                ind++;
                        }
                        mp[key_token] = val_token;
                }
        } else cout << "Couldn't open file " << argv[1] << ".\n";

        string query;
        while (cin >> query) {
                if (mp.find(query) == mp.end()) cout << "Not Found.\n";
                else cout << mp[query] << "\n";
        }
}
```

# Red-Black Tree

Applications in Linux Kernel’s completely fair scheduler

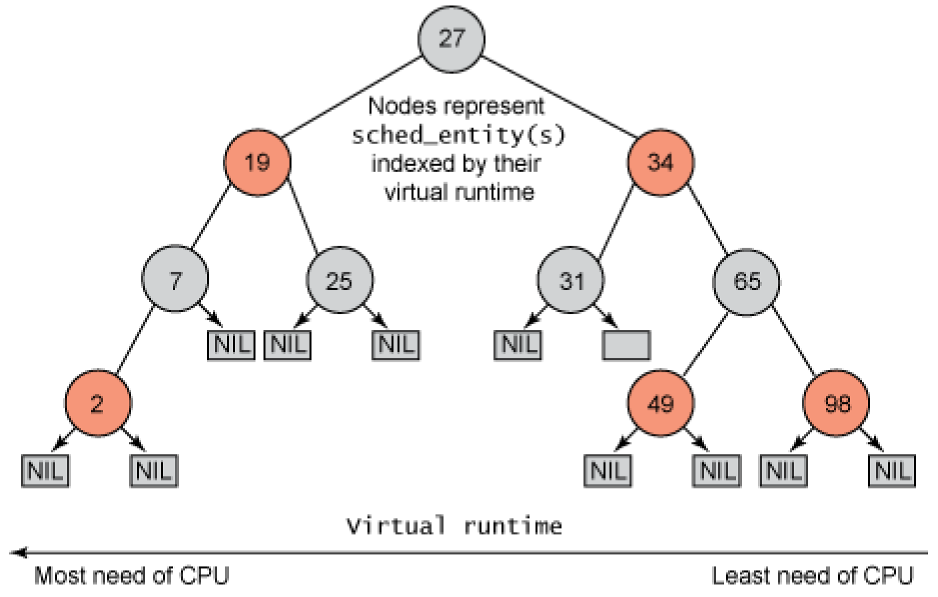
## Problem: OS Scheduling

- Goal: fairness in dividing processor time to tasks.
- Task scheduling is an important aspect to operating system design.
- Example Schedulers:
  - First Come, First Serve
  - Shortest Job First
  - Priority
  - Round Robin

## Completely Fair Scheduler

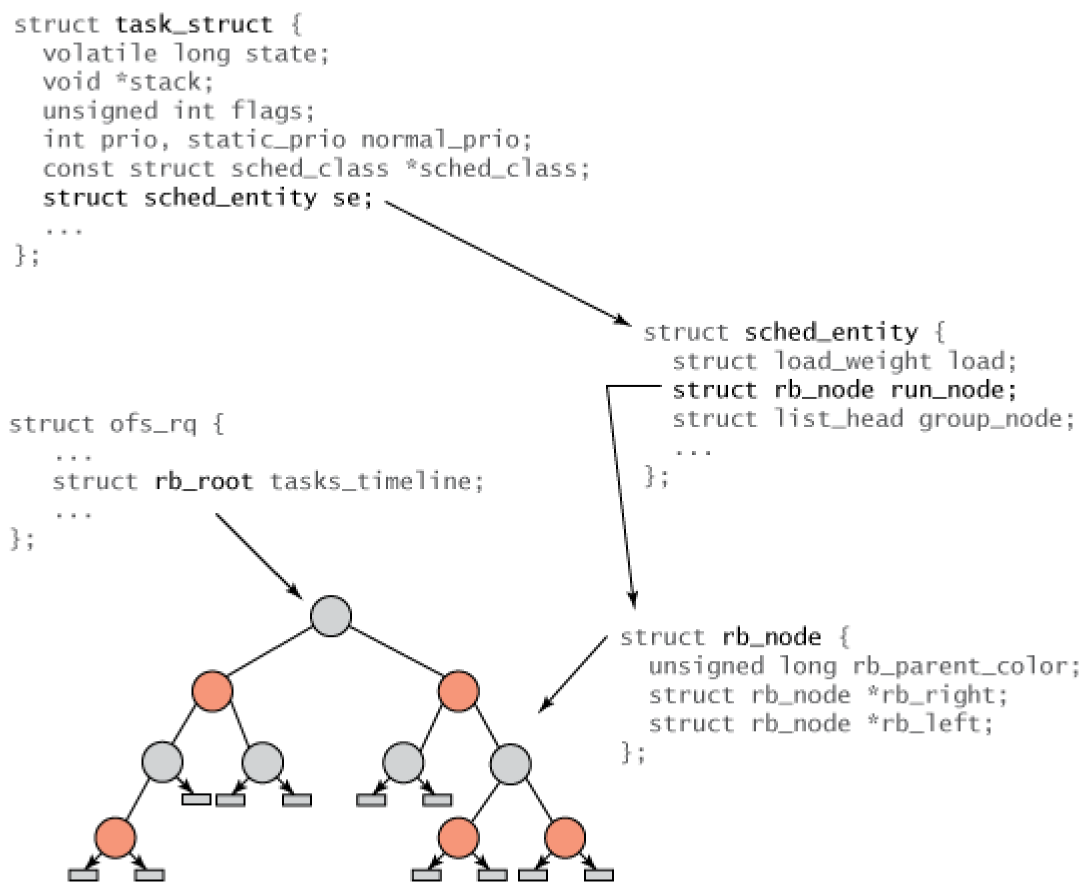
- Queue ordered in terms of “virtual run time”
  - smallest value picked for using CPU
  - small values: tasks have received less time on CPU
  - tasks blocked on I/O have smaller values
  - execution time on CPU added to value
  - priorities cause different decays of values
- Bottom Line: The smaller a task's virtual runtime, the higher its need for the processor

## Example of CFS Tree



- The key for each node is the virtual runtime of the corresponding task.
- To pick the next task to run, simply take the leftmost node.

# C++ Implementation



```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
};
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
};
```

```
struct ofs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```

```
struct rb_node {
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
```

- task_struct : All tasks within Linux.
- sched_entity : Entity created to track scheduling information
- rb_root : root of the tree
- Internal nodes represent one or more tasks that are runnable.
- The rb_node is contained within the sched_entity structure, which includes the rb_node reference, load weight, and a variety of statistics data
- sched_entity contains the vruntime

Based on “Inside the Linux 2.6 Completely Fair Scheduler”- http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/, Image Source – same.

# Tries

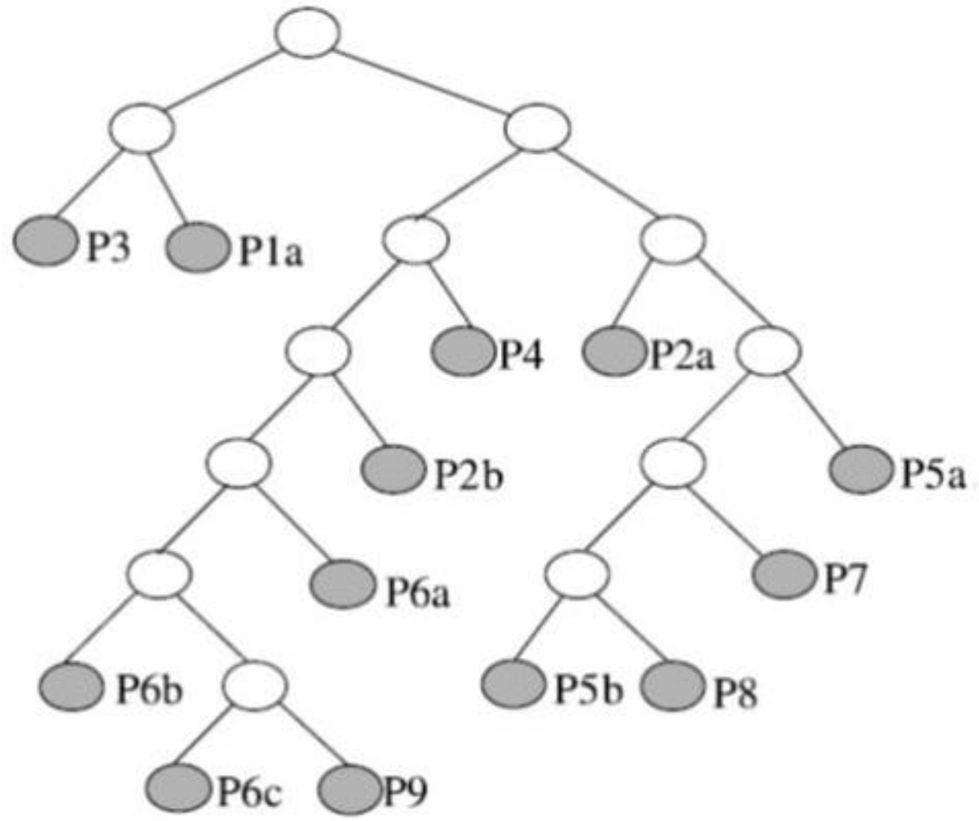
Applications in IP Lookup

# IP Lookup

- IP lookup is the most frequently used job. Searching IP from huge number of IPs can be done using prefix tries easily in O(32) for IPv4 and O(128) for IPv6
- **Route Lookup**: Each IP address lookup starts at the root node of the trie. Based on the destination address of the packet, the search procedure determines whether the left or the right node is to be visited. The prefix node found along the path that contains forwarding information is maintained as Best Matching Prefix (BMP) while the trie is traversed.

Prefix database

| | |
|---|---|
| P1 | * |
| P2 | 1* |
| P3 | 00* |
| P4 | 101* |
| P5 | 111* |
| P6 | 1000* |
| P7 | 11101* |
| P8 | 111001* |
| P9 | 1000011* |



# Queue

Applications in Round-Robin OS Scheduling

# Round Robin OS Scheduling

- **Round-Robin** (RR) is one of the algorithms employed by process and network schedulers in computing.
- It equally divides quanta of time among various process.

**Job 1 = Total time to complete 250 ms (quantum 100 ms)**.

First allocation = 100 ms.

Second allocation = 100 ms.

Third allocation = 100 ms but *job1* self-terminates after 50 ms.

Total CPU time of *job1* = 250 ms

# Complexity Analysis

| | | | |
|---|---|---|---|
| P0 | | | P0 arrives and gets processed |
| P0 | P1 | | P1 arrives and wait for the quanta to expire |
| P1 | P0 | | Quanta expires , P1 getting processed |
| P1 | P0 | P2 | P2 arrives |
| P0 | P2 | | P1 processed, jumps to P2 even quanta is left |
| P2 | | | P0 processed |

- Complexity: Insert O(1), Fetch O(1)
- Weighted round robin

# Heap

Applications in LRU Cache replacement policy

## LRU Replacement Policy

- LRU (least recently used) cache is a good way of implementing cache because it frees up precious memory based on the last time an element was used. (This reduces the misses and increases the performance)
- Whenever there is a cache miss (capacity miss) , we need to decide which block to replace from the cache such that the removed data doesn’t produce any further miss.

## Complexity Analysis

The data which is least recently used is removed. For this a min heap is used which tells the minimum used in O(1).

Whenever we access the block we update a counter which tells us how recently the element has been used.

- Update : O (log(n))

We read/write in a cache. Whenever there is a miss we bring the data from lower cache

- Delete in Cache : O(log(n))
- Insert in Cache: O(log(n))